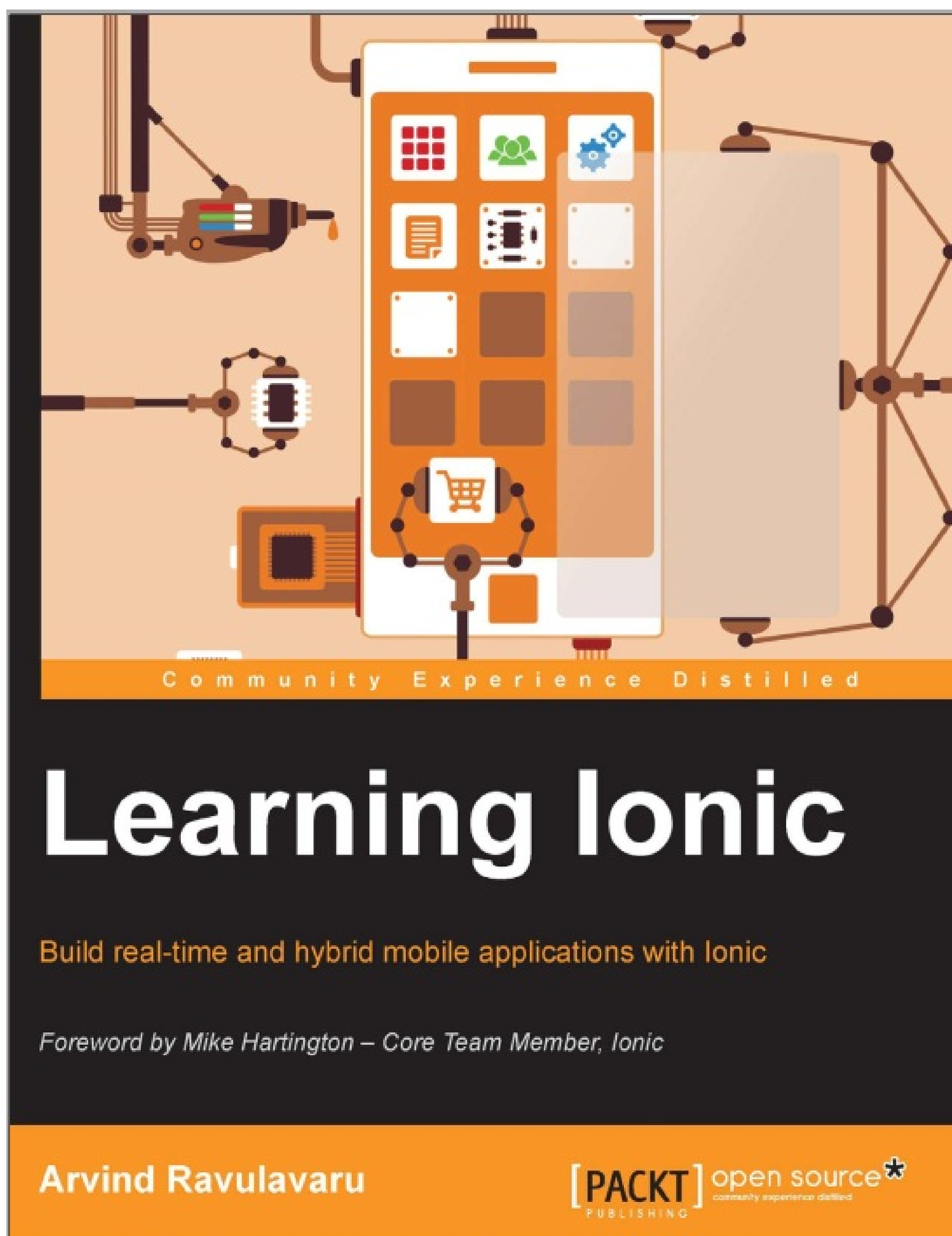
目錄

学习Ionic - 使用Ionic构建实时混合移动应用

作者:Arvind Ravulavaru



Ionic 1入门与进阶手册。

[Ionic 2入门与进阶](https://legacy.gitbook.com/book/adobeattheworld/building-mobile-apps-with-ionic-2/details\)

2016-10-7 首译完成
2016-10-18 开始review

2016-12-28 不更新了

有需要的可以自己去github fork

校对：ManInBoat

原本：https://github.com/learning-ionic

译本：https://github.com/AdoBeatTheWorld/learning_ionic_chinese

前言

本书是*Arvind Ravulavaru*数月辛勤劳作的结果，Arvind Ravulavaru是一个勤奋的开发者和作者，我和他之前有过多次愉快的合作经历。 本书是学习Ionic一个非常不错的选择，同时，对于有经验的开发者，本书也可以使你受益颇深。

Arvind带领大家走遍整个Ionic学习流程，从安装所需的软件开始，然后教会你如何设置本地SDK， 接着，Arvind带领大家一起学习Ionic的所有基础知识，例如Ionic的组件，使用UI-router导航，自定义样式，以及Ionic提供的API。 有了这些基本之后，作者带着大家制作了两个app：一个书店应用，和一个实时聊天应用。
对于有经验的开发者，本书展示了如何通过Cordova插件启用设备的本地API； 如何在AngularJS与语法中使用ngCordova（Ionic团队的另一个项目）和Cordova插件。
在聊天应用中，大家将会看到如何连接外部数据库，例如Firebase；也可以学习到在终端设备之间实时同步数据。

在作为Ionic的核心团队成员加入之前，我在另一家公司工作，在那里我制作了一些内部使用的混合app。 在经过对比所有的混合应用框架之后，我选择了Ionic，因为只有他提供了完整的混合移动开发解决方案。 我需要的所有东西Ionic都提供了。 这样一来，我就可以专注的制作应用，而不用专注于如何架构应用。（注：build 制作；architect 架构）。

Ionic为混合移动应用开发提供了一套完整的生态系统，为本地开发提供了一套低消耗，高效率的选择。 我们将在5月份发布Ionic的稳定版本，然后在夏天的时候发布3个平台服务的alpha版本。 我们放慢的打算；我们将会保持对Ionic SDK的开源的强力支持。 在Ionic，我和许多同事一起工作；同时我也在世界各地旅行进行Ionic布道；与Ionic的专家们一起研究代码。 我持续的被人们多么喜爱这个产品以及我们有这么一个动态的和积极的社区所感动。

你将会发现这是一本伟大的Ionic入门宝典。 这本书会带你领略更多的SDK知识。 谢谢你成为我们Ionic社区的一员。
好好享受吧！

Mike Hartington
Ionic核心团队成员

关于作者

Arvind Ravulavaru是一个有着超过6年软件开发经验的全栈专家。 最近2年，他广泛的涉猎了在服务端和客户端的JavaScript使用。 此前，Arvind的从业经历涵盖了大数据分析，云预防以及管弦乐编曲。 对于很多数据库他都有不同程度的研究，并使用HJJava和ASP.NET开发和架构了一个应用。

一年半前，Arvind开始启动了一个名为The Jackal of JavaScript的博客 **http://thejackalofjavascript.com**， 在这里，他写了很多关于使用纯JavaScript开发全栈应用的服务端和客户端的经验。 他写了很多不同的主题，包括：DNA分析，利用JavaScript进行情绪分析，使用JavaScript编写Raspberry Pi（树莓派？）， 和制作一个基于WebRTC的node-webkit视频聊天客户端。

此外，Arvind还为企业提供技术的培训与指导服务。 同时他还在进行创业，指导如何使用时下的强大的工具栈进行快速原型开发。 Arvind还指导如何将想法快速实现投放到市场。

他一直以不同的方式参与开源社区的贡献，让世界更好的去对待开发者。 作为一个咨询顾问，Arvind一直在尝试使用一些很棒的技术解决方案（语言无关）去实现那些精彩的想法，提升人类在进化链里的位置。

Arvind最近在Hyderabad创建他自己的公司。公司正在Internal of Things空间里制作自己的产品。 公司的目标是为世界制作价格合理的IoT产品，这样，每个人都可以享受科技带的生活改变。

想要了解Arvind的话，可以参考他的博客：http://thejackalofjavascript.com

Arvind同时也审阅了 使用*AngularJS*进行基于数据的开发，*Manoj Waikar*， *Packt*出版

前言

翻译了半个钟头的前言，因为没保存而被手误替换掉了，泪目....

版权

本书版权归PacktPub所有，有版权方面的问题请联系 **copyright@packtpub.com**

问题

问题反馈请联系 **questions@packtpub.com**

反馈

欢迎读者反馈，**feedback@packtpub.com**

如果在某些方面有专长并且对写作有兴趣或者想参与书籍，请阅读作者引导 **www.packtpub.com/authors**

源代码

源代码可以在 **http://www.packtpub.com**上面对应的书籍找到。

也可以在Github上查看源代码，提交issue，与作者互动:

https://www.github.com/learning-ionic

鸣谢

首先，我得感谢史蒂夫乔布斯，一直以来，他都是我的灵感和动力来源。 一次又一次的以神奇的价格将人们不需要的东西卖给他们并不是件很容易的事情：从Mac Book到Apple watch。

Rest In Peace， Mr. Jobs

我觉得简单的对我的家庭说一声谢谢是远远不够的，特别是我的母上大人。 在我低落和高涨的日子里，他们始终陪在我身边，分享我的快乐，在我低落的时候，鼓励我。 没有他们的鼓励和支持，我是不会走得这么远的。

特别感谢Nicholas Gault和Andrew Nuss，他们长久以来对我提供的知识，支持和鼓励。

谢谢Udaykanth Rapeti给我传授了很多金句："如果你真的想要成功的话，做一个电子版的吧"，同时也感谢他一直以来对我的支持。 特别感谢Karthik Naidu, Pavan Musti,以及我在Deloitte工作的团队。我们一起走过的时候，阳光明媚。

谢谢Ali Baig，自称为尤达大湿的人，一直以来和我分享他的建议和意见。让意见和建议来得更猛烈些吧！ 隆重感谢Accenture和Cactus团队的新血液们。我非常珍惜和你们一起学习，一起玩耍，一起rock的美好日子！ 前进吧，Cactus，加油！

再次小小的感谢一下我的初恋。你教会了我很多东西，让我成为一个更好的人。 和你分手是我专注技术的动力之一，也造就了现在的我。（注：要成功，先分手） 谢谢你。

9个章节和一个附录是一个很大的工程。编辑和审查也是。真诚的感谢Merwyn D'souza，本书的内容开发主编。 以及本书的审查人员Bramus Van Damme，Ian Pridham，以及Indermohan Singh。 由于他们的努力，本书得以更好的展现出来。作为本书的的技术主编，Shashank Desai，与他一起工作真的是一件很荣幸的事情。 我很高兴这是他在转换到另一项目经理之前的最后一本书。恭喜！ 同时，感谢Nikhil和其他Packt Publishing团队成员，让这本书得以出世。

特别感谢Hemal Desai找我写这本书。如果没有她，就没有这本书。

同时，一个大大的赞送给我的博客读者。你们好棒。

最后，致敬开发者社区的Robin Hoods，跟我开发和分享了他的免费的代码！

第一章 搭载AngularJS

Ionic是使用最广的移动混合框架之一。在写这篇文章的时候，Ionic的Github目录已经累计有17,000个star和2,700个fork。 Ionic建立在AngularJS之上，是一个用来制作MVW的史诗级的框架。 本章是一个介绍性的章节，主要是用来介绍AngularJS以及他是如何为Ionic提供强力支持的。 同时我们也会学习几个关键AngularJS组件，也就是在使用Ionic的时候会常用到的几个指令和服务。

> 本书会假设你对AngularJS有一个基本的认识。如果没有的话，建议你先看看《*AngularJS Essentials*》*Rodrigo Branas*或者 *Learning AngularJS, Jack Herrington*的视频，这些可以帮助你对AngularJS有个基本的认识。

本章只会解释AngularJS指令和服务。想要了解更多的AngularJS核心内容，参考上面提供的书和视频。

本章涵盖主题：

1. 什么是关注点分离
2. AngularJS是如何解决这个问题的
3. 什么AngularJS内建指令与自定义指令
4. 什么AngularJS服务，自定义服务是怎么工作的
5. Ionic里面是怎样去平衡指令与服务的

分节：

- 关注点分离 separation of concerns
- 指令 directives
- 服务 service
- 资源与总结

关注点分离

尽管服务端应用发展这么多年，Web演变至今，客户端驱动应用已经逐步超过服务端驱动应用。想当年，服务端告诉客户端该去干什么，客户端就乖乖的干什么。

随着异步调用与高交互网页的快速发展，使用客户端驱动要比服务端驱动能带来更好的用户体验。例如jQuery和Zepto这样的第三方库可以轻松帮助我们达到这个目标。

举一个经典的例子为证：用户在文本框里面输入数据，然后点击一个*Submit*按钮。之后这些数据将会经过*AJAX* **post**到服务端，无需刷新页面，UI使用服务端响应数据局部更新即可。

如果使用 *jQuery*（伪代码）的话，代码如下：

```
var textBox = $('#textbox');
var subBtn = $('#submitBtn');
subBtn.on('click', function(e) {
    e.preventDefault();
    var value = textbox.val().trim();
    if (!value) {
        alert('Please enter a value');
    return;
}

// 调用AJAX请求以获取服务端数据
var html2Render = '';
$.post('/getResults', {
    query: value
})
.done(functio\(data) {
    // 处理返回结果
    var results = data.results;
    for (var i = 0; i < results.length; i++) {
        // 为每条结果数据生成一个html元素标签
        var res = results[i];
        html2Render += ' < div class = "result" > ';
        html2Render += ' < h2  > ' + res.heading + ' < /h2 >';
        html2Render += ' < span > ' + res.summary + ' < /span >';
        html2Render += ' < a href = "' + res.link + '" < ' + res.linkText + ' < /a >';
        html2Render += ' < /div >'
    }

    // 将整个结果的html元素添加到页面上的容器里面
    $('#resultsContainer').html(html2Render);

    });
});
```

> 以上代码不可执行，仅供参考之用。

当按钮被点击的时候，文本框的值将被post到服务端。然后，前端将会根据服务端返回的结果（json对象）生成一个html标记，注入到结果容器里面。

但是，以上代码的可维护性如何？

怎样才能测试其中单独的代码片段呢？比如，测试数据的验证工作是否正常或者服务端响应是否正常。

假设我们要对结果页面进行更改（例如在每个结果页面之间添加一个favicon作为内联图标），那么在之前的代码里面导入这个变更的难易度又是怎样的呢？

这就是一个分离的关注点。这些分离点都是存在于数据验证，进行AJAX请求和构建html标记之间的。他们之间都是高度耦合的，如果其中一个break掉了，所有其他的都会break，上述所有代码就都无法重用了。

如果我们把上述代码分离成不同的组件，那么到最后我们将会得到一个**Model View Controller**（**MVC**）架构。在典型的MVC架构中，model是存放数据的地方，controller是在把数据展示到view之前对数据进行处理（原作：massage 马杀鸡，按摩）的地方。

与服务端MVC不同的是，客户端MVC有一个额外的组件名为router（路由器）。路由器一般是页面的URL，用来指示需要加载哪个model/view/controller。

这就是AngularJS背后的基本理念，也是他如何在提供单页面应用架构的同时分离了关注点。

在前面的示例中，服务端交互层（AJAX）将会从主代码中分离出来，然后根据需求与某个controller进行交互。

在理解了这些之后，我们现在将要快速学习几个AngularJS关键组件。

AngularJS 组件

与大部分客户端JavaScript框架不同的是，AngularJS是从HTML驱动的。 在一个标准的网页中， AngularJS负责在各个关键代码片之间进行连接。 所以，当你想要为你的HTML页面添加一些AngularJS指令并且包含AngularJS的时候，你可以不用写一行代码就能够轻松加愉快的搞定一个简单的应用。

为了证明我不是吹牛，我们将构建一个带验证功能的没有一行JavaScript代码的登录表单：

```
<html ng-app="">
<head>
    <script src="angular.min.js" type="text/JavaScript"></script>
<head>
    <body>
        <h1>Login Form</h1>
        <form name="form" method="POST" action="/authenticate">
            <label>Email Address</label>
            <input type="email" name="email" ng-model="email" required>
            <label>Password</label>
            <input type="password" name="password" ng-model="password" required>
            <input type="submit" ng-disabled="!email || !password"  value="Login">
        </form>
    </body>
</html>
```

上面的的代码片段中，以 *ng-* 开头的属性叫做AngularJS指令。

在这里， *ng-disable* 指令的职责是当输入的e-mail和password的值无效时启用**Submit**的disabled属性。

同时，指令的范围被安全的限制到了他声明的元件以及他的子元件的范围内。 这里解决了JavaScript语言的另一个关键性问题：如果没有正确的声明一个变量，那么他就会被附加到Global Object，也就是Window Object。

> 如果你对范围（scope，范围或者作用域）一无所知，那么建议你先浏览https://docs.angularjs.org/guide/scope 。 如果你对scope和root scope没有正确的认识的话，那么本书学起来将会非常的难。

现在，我们将会进行到下一个AngularJS组件，名为**Dependency Injection**（**DI**，依赖注入）。DI负责在有需求的的地方注入相应的代码片。这也是关注点分离的关键促成点之一。

你可以根据需求注入不同的AngularJS组件。例如，你可以在controller里面注入一个service。

> DI是AngularJS中另一个你必须了解的核心组件。关于DI的具体信息，可以查看：https://docs.angularjs.org/guide/di

为了便于理解service和controller，我们回退一步。在一个典型的客户端MVC框架中，我们知道model存放数据，view展示数据，controller处理model里面的数据并展示到view。

在AngularJS中，你可以这么理解上面的信息：

- HTML - Views
- AngularJS 控制器 - Controllers
- Scope 对象 - Model

在AngularJS中，HTML充当模块媒介的角色。AngularJS控制器从scope对象中或者服务的响应数据中取得数据，然后将它们组合成最终的view展示到网页上。 这和我们在搜索范例中做的非常类似，我们通过遍历结果，构建HTML字符串，然后将他注入到DOM中。 在这里，你可以看到，我们将功能分离到不同的组件中去了。 重申一下，HTML页面表演的角色是模板，factory组件负责发起AJAX请求。 最终，controller负责将factory的数据传递到view，view会生成实际UI。

AngularJS版本的搜索范例如下。

index.html 应该是这样的：

```
<html ng-app="searchApp">
    <head>
        <script src="angular.min.js" type="text/JavaScript">
        <script src="app.js" type="text/JavaScript">
    </head>
    <body ng-controller="AppCtrl">
        <h1>Search Page</h1>
        <form>
            <label>Search : </label>
            <input type="text" name="query" ng-model="query" required>
            <input type="button" ng-disabled="!query" value="Search" ng-click="search(
)">
        </form>
        <div ng-repeat="res in results">
        <h2>{{res.heading}}</h2>
        <span>{{res.summary}}</span>
        <a ng-href="{{res.link}}">{{res.linkText}}</a>
        </div>
    </body>
</html>
```

*app.js*是这样的：

```
var searchApp = angular.module('searchApp', []);
searchApp.factory('ResultsFactory', ['$http', function($http) {
    return {
        getResults : function(query){
            return $http.post('/getResults', query);
        }
    };
}]);

searchApp.controller('AppCtrl', ['$scope','ResultsFactory',function($scope, ResultsFac
tory) {
    $scope.query = '';
    $scope.results = [];
    $scope.search = function(){
        var q = {
            query : $scope.query
        };

        ResultsFactory.getResults(q).then(function(response){
        $scope.results = response.data.results;

        });
    }
}]);
```

> 在AngularJS中，factory组件和service组件是交互使用的。想要更好的理解，请参考stack overflow里面的一个讨论：http://stackoverflow.com/questions/15666048/angularjs-service-vs-provider-vs-factory

*index.html*是由HTML模板组成的。当页面加载时，模板默认（打死我也不会说缺省）是隐藏的。当结果数组发出数据的时候，模板将通过*ng-repeat*指令生成html标记。

在*app.js*中，我们先创建一个名为*searchApp*的AngularJS模组。然后我们创建了一个叫做*ResultsFactory*的工厂，这个工厂的唯一作用是发起AJAX请求返回一个promise。 最后，我们创建了一个controller，名为*AppCtrl*，用于与factory合作更新视图。

> 如果你对promise一无所知，参考：http://www.dwmkerr.com/promises-in-angularjs-the-definitive-guide/

按钮上*ng-click*指令声明的*search*在AppCtrl中已经设置好了。这个按钮只有在搜索框里面输入有效数据的时候才会启用。当点击**Search**按钮的时候，controller里面注册的事件监听器将会被调用。 这里我们将创建服务端需要的query对象以推进和调用*ResultFactory*里面的*getResults*方法。*getResults*方法返回一个promise，这个promise对象在服务端响应返回之后才可以被解析。 假设服务端响应成功，我们会把服务端返回的搜索结果传入*$scope.results*。

*$scope*对象的变更将触发*results*里面所有实例的更新。这样一来就触发了HTML模板上的*ng-repeat*指令，然后这个指令解析新的*results*数组，生成html标记。这样，UI上就成功的显示了最新的搜索结果。

> 下载范例代码或者咨询作者，请访问Github： Github地址:https://github.com/learning-ionic/Chapter-1

上面的范例给你展示了如何有序推进你的代码架构，使你的代码可维护，可测试。 现在，如果想要在每个搜索结果之间插入一个图片这样的需求就变得非常简单了。

AngularJS 指令

引用自AngularJS文档：

> "At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specifed behavior to that DOM element or even transform the DOM element and its children."
>
> 指令是一个高级别的DOM元素（例如一个属性，元素名，备注或者CSS类）的制作者，告诉AngularJS的HTML编译器（$compiler）给对应的DOM元素添加指定的行为或者转变DOM元素和他的子元素

当你想要在网页上抽离出公用功能的时候，这是一个非常有用的特性。AngularJ的指令已经达成了这些功能，例如：

- ng-app：在没有传入值的时候，这个指令初始化一个新的默认的AngularJS模块；反之，会初始化一个有名字的模块。
- ng-model： 将输入的元素的值映射到当前scope
- ng-show： 当传递给ng-show的表达式的值为true的时候，显示对应的DOM元素。
- ng-hide：当传给ng-hide的表达式的值为true的时候，隐藏对应的DOM元素。
- ng-repeat： 这条指令根据传入的表达式迭代当前标签以及他的子元素。

回想我们之前创建的Search App，想象一下应用中还有很多其他页面需要用到这个搜索表单。并且，他们期望的结果也差不多。
此时，我们可以将这个表单功能抽离成一个自定义的指令，而不是去复制粘贴controller和HTML模板代码。

然后你就可以在DOM元素的属性里用到一条新的指令，例如

，或者你可以创建你自己的标签或元件，例如</my-search>。

这样你只用写一遍搜索功能，然后就可以各处使用。AngularJS负责在进入view的时候初始化指令，在离开view的时候销毁指令。

我们将会为Search App创建一个名为*my-search*的新指令。 这条指令的唯一功能是渲染一个文本框和一个按钮。当点击**Search**按钮的时候，我们将会拿取结果然后将它们展示到搜索表单下面。

现在开工。

和其他AngularJS组件一样，所有的指令都绑定到一个模块。在咱们的案例中，我们已经有一个名叫searchApp的模块。我们将绑定一条新的指令到这个模块：

```
searchApp.directive('mySearch', [function () {
  return {
    template : 'This is Search template',
    restrict: 'E',
    link: function (scope, iElement, iAttrs) {

    }
  };
}]);
```

这条指令名为*mySearch*，命名方式的驼峰式。当在HTML中使用指条指令的时候，AngularJS将负责用*my-search*匹配这条指令。 我们将在*template*属性中设置一个范例文本。我们限制此指令用作为元素（E）类型。

> AngularJS里面其他可以限制的值是 A（attribute 属性，） C(class 类），以及 M（comment 备注）。你也可以允许指令使用所有的4个格式（ACEM）。

我们创建了一个链接方法。每当指令进入到view的时候，这个方法都会被调用。这个方法有以下3个参数注入其中：

- scope： 这个参数规定了这个标签在DOM里面的范围。例如，他可以在AppCtrl里面，甚至直接在rootScope(ng-app)里面。
- iElement： 指令附加到的宿主元素
- iAttrs： 当前元素的所有属性。

在我们的指令中，我们不会使用iAttrs，因为我们的*my-search*标签上没有属性。 在复杂的指令当中，最好的做法是将你的指令模板抽离成另一个文件，然后在指令中使用*templateUrl*来引用他。 接下来，我们就使用这种做法。
在*index.html*的同一目录下，新建一个文件名为*directive.html*，然后添加以下内容：

```
<form>
  <label>Search : </label>
  <input type="text" name="query" ng-model="query" required>;
  <input type="button" ng-disabled="!query" value="Search" ngclick="search()">
</form>
<div ng-repeat="res in results">
  <h2>{{res.heading}}</h2>
  <span>{{res.summary}}</span>
  <a ng-href="{{res.link}}">{{res.linkText}}</a>
</div>
```

就这么简单，我们在*index.html*使用这条指令替换相应的搜索代码。 现在，我们将会在指令内部给按钮注册一个*click*事件监听器。更新后的指令如下：

```
searchApp.directive('mySearch', [function() {
  return {
  templateUrl: './directive.html',
  restrict: 'E',
  link: function postLink(scope, iElement, iAttrs) {
    scope.search = function() {
      var q = {
        query : scope.query
      };

      //Interact with the factory (next step)
    }
  }
  };
}])
```

如上代码所示，当按钮的*click*事件触发之后执行了*scope.search*，*scope.query*返回了文本框的值。这跟我们在控制器里面做的差不多。

现在，当用户在输入一些文本然后点击**Search**按钮的时候，我们会调用*ResultFactory*里的*getResults*方法。然后，一旦结果返回，我们将会把他们绑定到scope的*results*属性上。 完整的指令代码如下：

```
searchApp.directive('mySearch', ['ResultsFactory',
function(ResultsFactory) {
  return {
    templateUrl: './directive.html',
    restrict: 'E',
    link: function postLink(scope, iElement, iAttrs) {
      scope.search = function() {
        var q = {
          query : scope.query
        };
        ResultsFactory.getResults(q).then(function(response){
          scope.results = response.data.results;
        });
      }
    }
  }
}])
```

有了这些，我们可以将index.html更新为：

```
<html ng-app="searchApp">
<head>
  <script src="angular.min.js" type="text/JavaScript"></script>
  <script src="app.js" type="text/JavaScript"></script>
</head>
<body>
  <my-search></my-search>
</body>
</html>
```

然后app.js更新为以下：

```
var searchApp = angular.module('searchApp', []);
searchApp.factory('ResultsFactory', ['$http', function($http){
    return {
      getResults : function(query){
      return $http.post('/getResults', query);
    }
  };
}]);
searchApp.directive('mySearch', ['ResultsFactory',function(ResultsFactory) {
  return {
    templateUrl: './directive.html',
    restrict: 'E',
    link: function postLink(scope, iElement, iAttrs) {
      scope.search = function() {
        var q = {
          query : scope.query
        };
        ResultsFactory.getResults(q).
          then(function(response){
            scope.results = response.data.results;
        });
      }
    }
  };
}]);
```

异常简单，但是非常强大！
现在，当你在哪里需要一个搜索条的时候，直接扔一个</my-search>就可以了。
你也可以给这条指令进行升级：给他传入一个名为*results-target*的属性。这实际上是页面上的某元素的ID。 因此，你可以在这个提供的元素里面显示结果，而不是像之前那样在搜索条下面显示结果。

> AngularJS自带一个轻量版本的jQuery库名为jqLite。 jqLite不支持选择器查询。如果你想要用jQuery替代jqLite的话，那么需要在AngularJS之前添加jQuery。 更多关于jqLite的信息请参考：https://docs.angularjs.org/api/ng/function/angular.element

在处理DOM元素的时候，这个特殊功能使得AngularJS指令成为组件重用的完美解决方案。

此时，当你需要给你的Ionic app添加一个新的导航条的时候，你只需要扔一个**ion-nav-bar**标签进去就可以了，如下：

```
<ion-nav-bar class="bar-positive">
  <ion-nav-back-button>
  </ion-nav-back-button>
</ion-nav-bar>
```

然后，一切将会尘埃落定。
当我们回首理解自定义指令的痛苦的时候，你将会很容易的的联想到所有由AngularJS指令所创建的Ionic组件。

AngularJS 服务

> 用到的名词：
>
> - directives 指令
> - controller 控制器
> - Dependency Injection 依赖注入
> - lazy initialize 延迟初始化
> - singlton 单例
> - destroy 销毁
> - factory 工厂
> - local storage 本地存储

AngularJS服务都可以通过DI注入到指令与控制器里。
这些对象是由一系列的简单业务逻辑碎片组成。

当组件需要依赖他们的时候，AngularJS服务都会延迟初始化。同时，所有的服务都是单例，单例的意思是在每个app里面仅初始化一次。
这样可以让服务在控制器之间完美的共享数据，并根据需求将他们存入缓存。

*$interval*是一个AngularJS服务。*$interval*和*setTimeInterval()*是一样的。 在注册这个服务的时候，他将*setTimeInterval()*包装起来并且返回一个*promise*对象。 这个promise后续可以用来销毁*$interval*。

另一个比较简单的服务是*$log*。这个服务将日志信息输出到浏览器控制台。来个比较简单的例子：

```
myApp.controller('logCtrl', ['$log', function($log) {
    $log.log('Log Ctrl Initialized');
}]);
```

由此，可以看到服务的强大之处以及理解业务逻辑碎片是如何在整个app中重用的。

你也可以自定义自己的服务。例如建一个计算器服务试试看，包括了加，减，乘，除，平方等等方法。

回到我们的搜索App，我们使用了一个工厂负责服务端通讯。现在，我们将添加我们自己的服务。

> 服务与工厂组件之间是可以互换的。可以参考这里:
> http://stackoverflow.com/questions/15666048/servicevs-provider-vs-factory

例如，当用户搜索指定关键字然后展示搜索结果的时候，我们比较倾向于将结果缓存中本地。 这样可以保证下次用户搜索同一个关键字的时候，我们无需发起AJAX请求，而是直接展示本地存储中的数据（和离线模式类似）。

我们就这么设计我们的服务。我们的服务将包含以下三个方法：

- isLocalStorageAvailiable():这个方法检查浏览器是否支持本地存储API
- saveSearchResult(keyword, searchResult): 这个方法将关键字和搜索结果存放到本地存储
- isResultPresent(keyword):这个方法返回指定关键字的搜索结果

我们的服务大概是这样的：

```
searchApp.service('LocalStorageAPI', [function() {
    this.isLocalStorageAvailable = function() {
        return (typeof(localStorage) !== "undefined");
    };
    this.saveSearchResult = function(keyword, searchResult) {
        return localStorage.setItem(keyword, JSON.stringify(searchResult));
    };
    this.isResultPresent = function(keyword) {
        return JSON.parse(localStorage.getItem(keyword));
    };
}]);
```

> 本地存储不能存储对象。因此，我们必须在缓存对象之前将其字符串化，在获得本地缓存的时候对象化。

现在，当我们处理搜索的时候，我们的指令将会使用这个服务。更新后的mySearch指令是这样子的：

```
searchApp.directive('mySearch', ['ResultsFactory', 'LocalStorageAPI',
    function(ResultsFactory, LocalStorageAPI) {
        return {
        templateUrl: './directive.html',
        restrict: 'E',
        link: function postLink(scope, iElement, iAttrs) {
            var lsAvailable = LocalStorageAPI.isLocalStorageAvailable();
            scope.search = function() {
                if (lsAvailable) {
                var results = LocalStorageAPI.isResultPresent(scope.query);
                    if (results) {
                        scope.results = results;
                        return;
                    }
                }
                var q = {
                    query: scope.query
                };
                ResultsFactory.getResults(q).
                then(function(response) {
                    scope.results = response.data.results;
                    if (lsAvailable) {
                        LocalStorageAPI.saveSearchResult(scope.query, data.data.result
s);
                    }
                });
            }
            }
        };
}]);
```

之前说过，我们会先检查本地存储是否可用，然后使用*LocalStorageAPI*服务进行数据的存与取。

同样，Ionic也提供了自定义的服务供我们去消化。详细信息我们可以去 第五章 - *Ionic*指令与服务 查看。
下面这个例子是Ionic的加载服务。这个服务展示了一个加载条和你提供的文本。例子代码大概是这样子的：

```
$ionicLoading.show({
    template: 'Loading...'
});
```

然后你就可以看到一个覆盖层（overlay）展示后台活动并阻断用户操作。

AngularJS 资源

> 用到的名词：
>
> - service 服务
> - factory 工厂
> - piece of code 代码片
> - HTML element html元素
> - mobile hybrid app 移动混合应用

以下推荐一些非常有价值的Github资源，看完他们你就会知道AngularJS世界里最强与最新的东西是什么:

- jmcunningham/AngularJS-Learning https://github.com/jmcunningham/AngularJS-Learning
- gianarb/awesome-angularjs https://github.com/gianarb/awesomeangularjs
- aruzmeister/awesome-angular https://github.com/aruzmeister/awesome-angular

> 注意：本书使用的是Ionic 1.0.0，AngularJS的版本是1.3.13

总结

在本章中我们看到了AngularJS是如何设计以达成关注点分离。 然后快速学习了我们在Ionic开发中将要用到的一些关键的AngularJS组件。 也学习了如何创建自定义服务和指令以及他们的使用。 在使用AngularJS的时候，创建可重用的代码片的首要原则是： 当处理HTML元素（DOM）的时候，使用指令；其他情况下用服务或者工厂。

你也理解了Ionic是如何使用AngularJS元件暴露出来的简单的易用的API去创建移动混合应用。

下一章里面，我们将会介绍Ionic。 你将学会如何设置他们，如何利用他创建一个基本的应用，以及理解项目架构。 同时也会带你领略一下开发移动混合应用的大观。

欢迎来到Ionic

在上一章中，我们学习了一些AngularJS的关键特性，也就是指令和服务。 在本章中，我们将探索移动混合应用的蓝图，安装开发Ionic应用必需的软件，最终搭建一些应用。

本章涵盖主题如下：

- 移动混合架构
- 什么是Aphache Cordova
- 什么是Ionic
- 安装开发和运行Ionic应用需要的工具
- 使用Ionic模块进行工作
- Yeoman Ionic Generator

> 用到的名词
>
> - Mobile Hybrid platform 移动混合平台
> - apps/application 应用
> - package 封包
> - installer 安装包
> - Hybrid App 混合应用

移动混合架构

在使用Ionic工作之前，我们需要了解一下移动混合平台的形势。

这个概念非常简单。 几乎每一个移动操作系统（使用Cordova的时候也叫做平台）都有开发应用的API。 这个API组成了一个叫做Web View的组件。一个Web View基本就是运行在一个移动应用里面的浏览器。 这个浏览器运行HTML,CSS，以及JS代码。这意味这你可以使用前面的技术创建网页，然后在你的应用里面执行。

你可以使用web开发的知识来开发本地-混合移动应用（此处的本地的意思是在应用与他的素材打包后安装为平台特定的格式文件），例如：

- Android 使用 Android Application Package(.apk)
- iOS使用iPhone Application Archive(.ipa)
- Windows Phone使用Application Package(.xap)

封包/安装包是由一些本地代码片组成，这些代码片负责初始化网页和其他显示网页内容所需的所有素材。

这一系列展示移动应用容器里面的你的应用业务逻辑的网页的东西，就叫做混合应用。

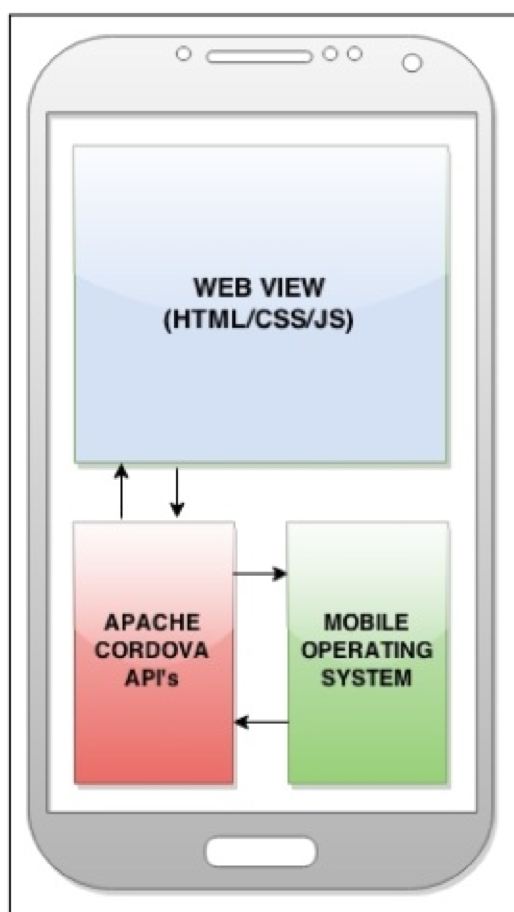
什么是Apache Cordova

简单来讲，Cordova是将你的网页应用与本地应用缝合的软件。
Apache Cordova上面是这么声明的：

> Apache Cordova is a platform for building native mobile applications using HTML, CSS and JavaScript. Apache Cordova是一个使用HTML,CSS,JavaScript制作本地移动应用的平台。

Apache Cordova不止是缝合网页应用和本地应用那么简单，他同时也提供了一套JavaScript写的API用来与设备的本地功能进行交互。 是的，你可以通过JavaScript调用你的摄像头，照相，然后发送到e-mail。屌不屌？

为了让你理解这其中其中缘由，我们可以看一下下面这张图：



可以看到，我们有一个执行HTML/CSS/JS代码的web view。这些代码可以只是简单独立的用户界面片段； 可以说向远程服务器请求数据而发送的AJAX请求。甚至，这些代码可以做更多的事情，例如直接跟蓝牙对话，取得附近的蓝牙设备列表。

> 关于本章，你可以通过以下Github地址获取源代码，发起issue，和作者沟通https://github.com/learning-ionic/Chapter-2

在后续的用例当中，Cordova有一系列的使用的JavaScript 与web view交互然后与设备本地语言（例如，Android的Java）交流， 从而提供在他们之间建立桥梁的API。例如，如果你想知道你的app中的JavaScript是运行在何种设备上的，你只需要在你的JS文件里面使用以下代码就可以实现：

```
var platform = device.platform;
```

安装好设备插件之后，你还可以通过JavaScript访问UUID，model，OS版本，以及设备内部web view使用的Cordova版本： ··· var uuid = device.uuid; var model = device.model; var version = device.version; var Cordova = device.Cordova; ···

后续第七章，Corfova和ngCordova我们会使用更多的Cordova插件的。

前面的解说是为了给你一个大概的了解，移动混合应用是如何构建的，你在web view里面如何通过JavaScript使用设备功能的。

> Cordova不会将HTML，CSS,以及JS代码转换为系统指定的二进制代码。他做的工作是包装HTML,CSS,JS代码然后在web view里面执行他们。

那么，你现在可能开始想到了，Ionic是 将创建好**HTML/CSS/JS**代码运行在**web view**里面并且与**Cordova**对话以访问设备指定**API**的 一个框架。

> 用到的名词
>
> - Command Line Interface (CLI) 命令行界面

什么是Ionic

Ionic是一个漂亮的，开源的，前端SDK，它使用HTML开发混合移动应用。 为了高交互应用，Ionic提供了为移动应用优化过的的HTML，CSS，以及JS组件，同时还有优化过的手势和工具。

与本联盟的其他框架对比，在Ionic使用了最小化DOM操作和硬件加速过度之后，他的执行效率很可观。 Ionic使用AngularJS作为他的JavaScript框架。

如Ionic这般拥有AngularJS的强力的框架，一切变得皆有可能（你可以在Ionic里面使用任何AngularJS组件）。 Ionic和Cordova设备API有完美整合。这意味着，你可以使用例如ngCordova这样的的库去访问设备API然后将他与Ionic美丽的用户界面组件整合起来。 Ionic有他自己的命令行界面，可以用作搭建，开发以及部署Ionic应用。 在使用Ionic CLI工作之前，我们需要安装一些软件。

软件安装

现在我们要开始安装运行Ionic应用所必需的软件了。

安装Node.js

由于Ionic的命令行工具和构建任务需要用到Node.js，那么我们第一个需要安装的就是他了：

1. 导航至网站 https://nodejs.org/
2. 点击主页上的安装按钮就会自动下载你当前操作系统对于的安装包。或者可以导航到 https://nodejs.org/download 然后下载指定的副本。
3. 直接执行安装包就可以安装Node.js了

为了验证Node.js是否安装成功了，打开一个Terminal终端（*nix 系统）或者 Prompt（Windows 系统）然后运行如下命令：

```
node -v
```

安装成功的话你就可以看到Node.js的版本了。接下来执行这个命令：

```
npm -v
```

你应该会看到npm的版本：

```
➜  ~  node -v
v0.12.2
➜  ~  npm -v
2.10.0
```

npm是Node Package Manager（Node包管理器），我们将使用他来为我们的Ionic项目下载不同的依赖包。

> 你只是在开发期间需要Node.js。上面图片显示的版本号只是为了展示正确的输出。你自己的版本有可能跟这里的版本一样，或者更新一些。 本章其他版本号的显示也是同理。

安装Git

Git是一个开源的免费版本控制系统。在我们的案例中，我们会使用一个名为Bower的包管理器，这个管理器就是用Git下载需要的库的。 同时Ionic CLI也是使用Git下载项目模板的。

导航至 *http://git-scm.com/downloads* ，下载对应平台的安装包，然后就可以安装了。 一旦你完成了安装，你就可以去你的终端/命令行运行如下命令：

```
git  --version
```

你将会看到如下输出：

```
→  ~  git --version
git version 2.3.2 (Apple Git-55)
```

安装Bower

我们将会使用Bower来管理我们应用的依赖库。Bower是一个与npm一样的包管理器，只是过他liner/flat（线性/平面）。 这种包管理器更适合去下载网页开发所需的素材。
Bower是建立在Node.js之上的。为了全局安装Bower，在终端/命令行里输入：

```
npm install -g bower
```

> 如果你需要sudo来运行上面的命令，请重新检查你的npm安装。 想要忽略sudo进行npm全局安装，请参考：http://competa.com/blog/2014/12/how-to-run-npm-without-sudo/

一旦Bower安装成功，你可以同样通过以下口令去验证：

```
bower -v
```

```
→  ~  bower -v
1.4.1
```

安装Gulp

接下来，我们将要安装Gulp，这是一个基于Node.js的构建系统。 他会将很多单调无聊的任务自动化处理。
例如，当你的网络项目准备好的时候，你可能想要压缩CSS，JS和HTML文件，为Web进行图片优化，将代码推送到你的生产环境； 在此情景下，Gulp就是你需要的工具。
Gulp有很多插件用来自动处理你大部分的单调的任务，他主要得益与开源社区的驱动。 在Ionic中，我们主要用Gulp来将SCSS代码转换成CSS。我们利用SCSS代码定制Ionic视觉元素。 在第四章 *Ionic与SCSS*中，我们会了解更多此方面的事宜。
为能全局安装gulp，运行如下命令：

```
npm install gulp -g
```

对于 *nix系统，运行这个命令：

```
sudo npm install gulp -g
```

一旦Gulp成功安装，我们同样可以使用以下命令去验证：

```
gulp -v
```

```
→  ~  gulp -v
[22:17:58] CLI version 3.8.11
```

安装Sublime Text

这完全是个可选的安装。因为每个人都有他自己喜欢的文本编辑器。在用了一些其他的文本编辑器之后，我深深的爱上了Sublime Text， 纯粹是因为他的简单，还有很多的Plug和Play包。

> 如果你想试试这个编辑器，请导航至 http://www.sublimetext.com/3 下载Sublime Text 3

安装Cordova和Ionic CLI

最后，为了完成Ionic安装，我们需要安装Ionic CLI。Ionic CLI是一个包装了Cordova CLI以及一些额外功能的包装。

> 本书中所有范例代码使用的是Cordova 5.0.0， Ionic CLI版本1.5.0，Ionic版本1.0.0（uranium-unicorn）

运行一下命令就可以安装Ionic CLI了：

```
npm install cordova@5.0.0 ionic@1.5.0 -g
```

验证安装是否成功可以运行如下命令：

```
cordova -v
```

同时也可以用这个命令：

```
ionic -v
```

```
➜  ~  cordova -v
5.0.0
➜  ~  ionic -v
1.5.0
```

如果好奇Ionic CLI里面有些什么，试试这个命令:

```
ionic
```

你将会看到这么一对任务：

```
➜  ~  ionic
                                     CLI v1.5.0
Usage: ionic task args

=======================

Available tasks: (use --help or -h for more info)

   start  ..........  Starts a new Ionic project in the specified PATH
   serve  ..........  Start a local development server for app dev/testing
   platform  .......  Add platform target for building an Ionic app
   run  ............  Run an Ionic project on a connected device
   emulate  ........  Emulate an Ionic project on a simulator or emulator
   build  ..........  Locally build an Ionic project for a given platform
   plugin  .........  Add a Cordova plugin
   resources  ......  Automatically create icon and splash screen resources (beta)
                      Put your images in the ./resources directory, named splash or icon.
                      Accepted file types are .png, .ai, and .psd.
                      Icons should be 192x192 px without rounded corners.
                      Splashscreens should be 2208x2208 px, with the image centered in the middle.
```

> 上面的截屏由于尺寸问题显示不完全，ionic还有其他一些任务

你可以阅读以下每个任务的解释以了解他们分别是做什么的。需要注意的是，其中有些任务时至今日仍是beta（试用）状态。

做完以上这些事情，我们已经安装好所有Ionic开发需要的软件了。

平台指引

在本书的最后，我们将会创建好app可以部署到设备上去。 由于Cordova使用HTML,CSS,以及JS代码作为输入和生成平台指定安装包，你需要在你的机器上准备好构建环境。

> Android用户可以按照Android Platform Guide：http://cordova.apache.org/docs/en/edge/guide_platforms_android_index.md.html#Android%20Platform%20Guide 的指引在你的本机上设置SDK。
> iOS用户可以按照iOS Platform Guide：http://cordova.apache.org/docs/en/edge/guide_platforms_ios_index.md.html#iOS%20Platform%20Guide 的指引在你的本机上设置SDK。 开发iOS应用必需要OSX环境。

直至今日，Ionic只至此Android 4.0+（虽然他在2.3上也可以运行）和iOS 6+版本的移动平台。但是Cordova支持的更多一点。

> Cordova支持的平台参考这里：http://cordova.apache.org/docs/en/edge/guide_platforms_index.md.html#Platform%20Guides

> 用到的名词：
>
> - Single Page Application (SPA) 单页面应用
> - dependency/dependencies 依赖库
> - bundle 包
> - template 模板
> - scaffold 搭建（如果更好的建议，请联系我）

Hello Ionic

现在咱们完成了软件安装的部分，我们将搭建一些Ionic app。
Ionic有三个可用模板：

- Blank： 空的Ionic项目，有一个页面
- Tabs： 使用Ionic tabs构建的一个范例应用
- Side Menu： 这个是一个侧边菜单驱动导航的范例应用 为了便于理解如何搭建Ionic app，我们从空白模板（Blank template）开始。
  为了保持我们的学习进程清晰明了，我们将创建一个文件夹作为Ionic项目来工作。创建一个文件夹名为*ionicApp*， 然后在里面创建一个文件夹名为*chapter2*。
  接下来，打开一个新的命令行/终端，进入到*ionicApp*目录下的*chapter2*。然后执行如下命令：

  ```
  ionic start -a "Example 1" -i app.example.one example1 blank
  ```

  以上命令中：
- -a "Example 1"：这是供凡人识别的app名字
- -i app.example.on：这是app的ID/域名倒转
- example1：这个是文件夹的名字
- blank：模板名
  > 更多Ionic start任务信息，请参考附录，更多主题与贴士

在执行任务的时候，Ionic CLI非常详细。这点你可以通过命令行/终端看得到，因为开始创建项目的时候里面会打印出大量的信息。
开始之后，会下载一个新的空白项目然后保存到example 文件夹。接下来，会从ionic-app-base的GitHub目录下载ionic-app-base， 在这之后， 会从ionic-starter-template的GitHub目录下载 ionic-starter-template。

之后，会将config文件里面的app名字和ID更新。接下来，会运行一段脚本然后下载5个Cordova插件：

- org.apache.cordova.device (https://gitHub.com/apache/cordovaplugin-device):这个是用来获取设备信息，我们在之前的章节已经看到过

- org.apache.cordova.console (https://gitHub.com/apache/cordovaplugin-console): 这个插件的用处是确保*console.log*（）有用
- cordova-plugin-whitelist (https://github.com/apache/cordovaplugin-whitelist): 这个插件实现了白名单政策，用来在Cordova 4.0的应用的web view中进行导航。
- cordova-plugin-splashscreen (https://github.com/apache/cordovaplugin-splashscreen): 这个插件在应用启动期间实现闪屏的展示与隐藏。
- com.ionic.keyboard (https://gitHub.com/driftyco/ionic-pluginskeyboard):这个插件提供更容易的键盘交互功能，在键盘隐藏/显示的时候发出事件。

所有这些信息稍后都会添加到*package.json*文件里面，然后一个暂时的*ionic.project*诞生了。一旦项目创建成功，你将会看到一系列的指引告诉你后续如何操作。 输出信息大概是这么个效果：

```
Your Ionic project is ready to go! Some quick tips:

 * cd into your project: $ cd example1

 * Setup this project to use Sass: ionic setup sass

 * Develop in the browser with live reload: ionic serve

 * Add a platform (ios or Android): ionic platform add ios [android]
   Note: iOS development requires OS X currently
   See the Android Platform Guide for full Android installation instructions:
   https://cordova.apache.org/docs/en/edge/guide_platforms_android_index.md.html

 * Build your app: ionic build <PLATFORM>

 * Simulate your app: ionic emulate <PLATFORM>

 * Run your app on a device: ionic run <PLATFORM>

 * Package an app using Ionic package service: ionic package <MODE> <PLATFORM>

For more help use ionic --help or ionic docs

Visit the Ionic docs: http://ionicframework.com/docs


New! Add push notifications to your Ionic app with Ionic Push (alpha)!
https://apps.ionic.io/signup
+---------------------------------------------------------+
+ New Ionic Updates for June 2015
+
+ The View App just landed. Preview your apps on any device
+ http://view.ionic.io
+
+ Invite anyone to preview and test your app
+ ionic share EMAIL
+
+ Generate splash screens and icons with ionic resource
+ http://ionicframework.com/blog/automating-icons-and-splash-screens/
+
+---------------------------------------------------------+
```

为了继续深入项目，我们将使用*cd*命令进入到*example1*文件夹。我们不按照终端/命令行的指引走，因为我们始终了解项目的设置。 一旦我们熟悉了Ionic的多样化组件，我们可以开始使用终端/命令行的指引了。

进入到*example1*目录之后，我们可以通过以下命令为app服务了：

```
ionic serve
```

这个命令将在端口*8100*上启动一个*dev*服务器，然后在你的默认（谁敢说缺省砍死谁）浏览器中启动app。 由于你需要使用Ionic，我个人（是作者不是译者）强烈推荐使用Google Chrome浏览器或者Mozilla Firefox。
当浏览器启动后，你可以看到空白模板。
如果此时你运行这个：

```
ionic serve
```

你将会看到显示了以下错误信息：

```
→ example1  ionic serve
The port 35729 was taken on the host localhost - using port 35730 instead
Running live reload server: http://localhost:35730
Watching : [ 'www/**/*', '!www/lib/**/*' ]
Running dev server: http://localhost:8100
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit

ionic $ An uncaught exception occured and has been reported to Ionic



listen EADDRINUSE (CLI v1.5.0)

Your system information:

Cordova CLI: 5.0.0
Gulp version:  CLI version 3.8.11
Gulp local:
Ionic Version: 1.0.0
Ionic CLI Version: 1.5.0
Ionic App Lib Version: 0.1.0
ios-deploy version: 1.7.0
ios-sim version: 3.1.1
OS: Mac OS X Yosemite
Node Version: v0.12.2
Xcode version: Xcode 6.3.2 Build version 6D2105
```

这个错误信息的意思是你本机的其他应用正在使用端口8100.想要解决这个问题，你可以在运行Ionic serve命令的时候指定其他端口，例如 *8200*：

```
ionic serve -p 8200
```

应用成功启动后，我们看到了浏览器中的输出，我们在终端/命令行可以看到如下信息：

```
→  example1  ionic serve
Running live reload server: http://localhost:35729
Watching : [ 'www/**/*', '!www/lib/**/*' ]
Running dev server: http://localhost:8100
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit

ionic $
```

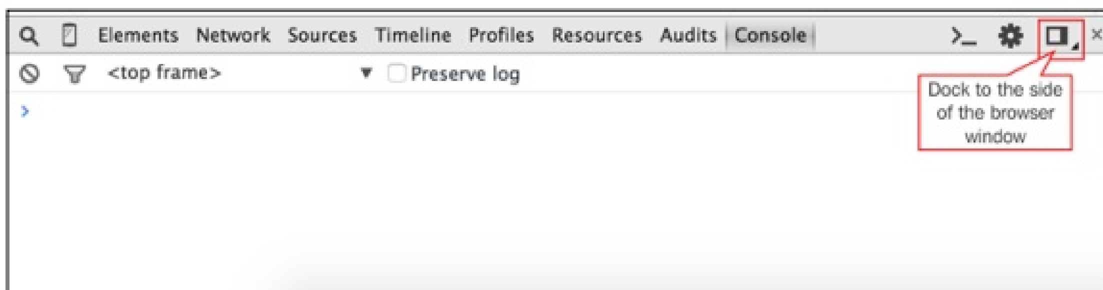
如前所述，Ionic CLI的任务非常详细。他不会让你闲着。在Ionic serve命令运行的时候你就可以看到了，你可以输入**R**然后回车，此时应用会重启。 同样，你可以按**C**来启用或者急用浏览JavaScript打印日志到终端/命令行。
运行完应用之后，按**Q**然后回车可以停止服务器。在键盘上按下**Ctrl + C**也是一样的。

浏览器开发者工具设置

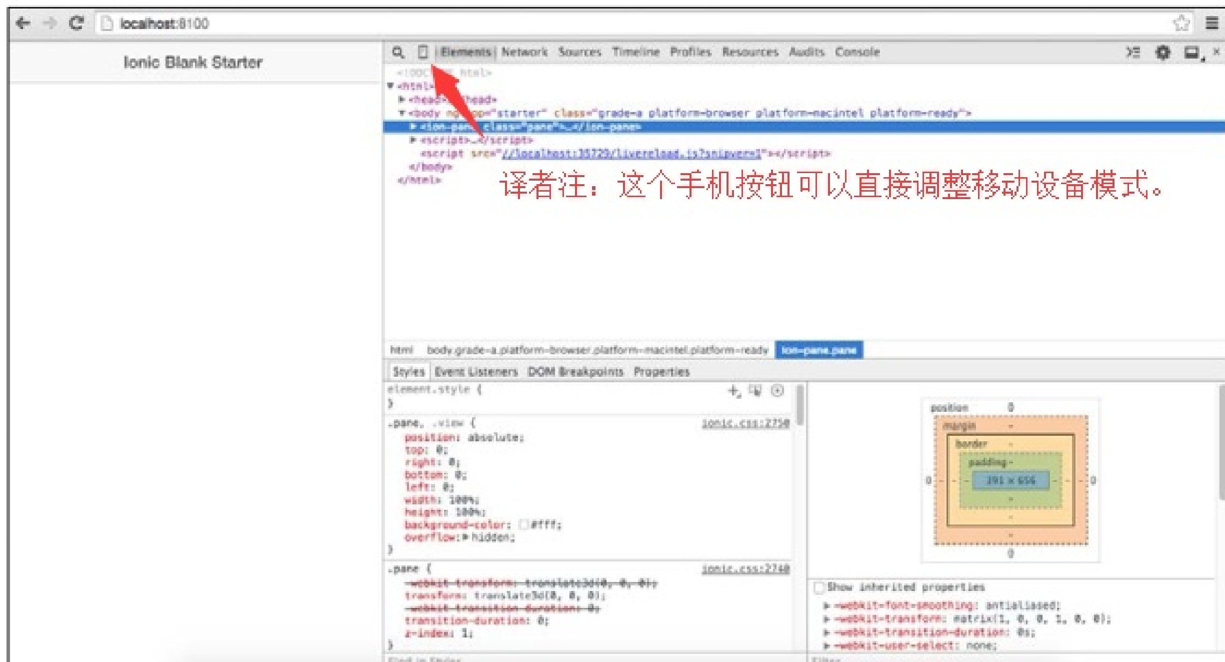
在我们深入之前，我建议使用以下的方式去设置好你的浏览器的开发者工具。

Google Chrome

Ionic应用启动后，通过*Cmd + Opt +I*（Mac）和 *Ctrl + Shift + I*（Windows/Linux）打开开发者工具。 然后点击顶行最后一个图标，也就是关闭按钮旁边那个，如下：



这个操作将会把开发者工具码到当前也的右边。拖动浏览器与开发者工具之间的分界条知道视图看起来像一个移动设备。如下：

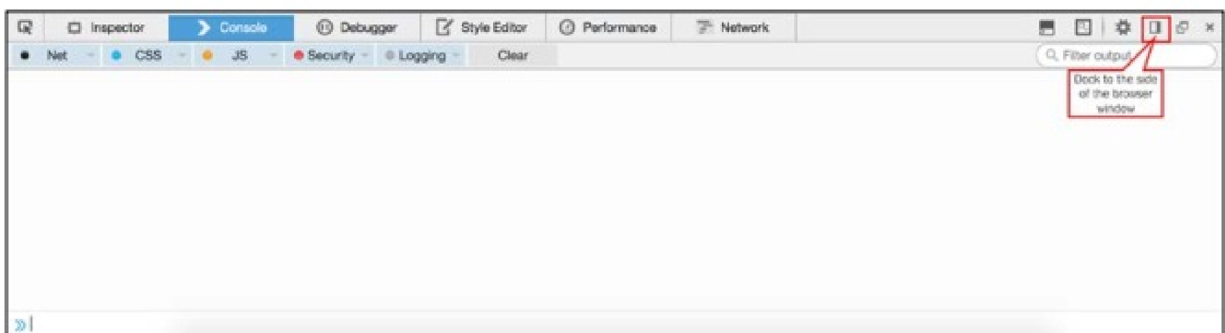


（图中红色信息为译者标注）

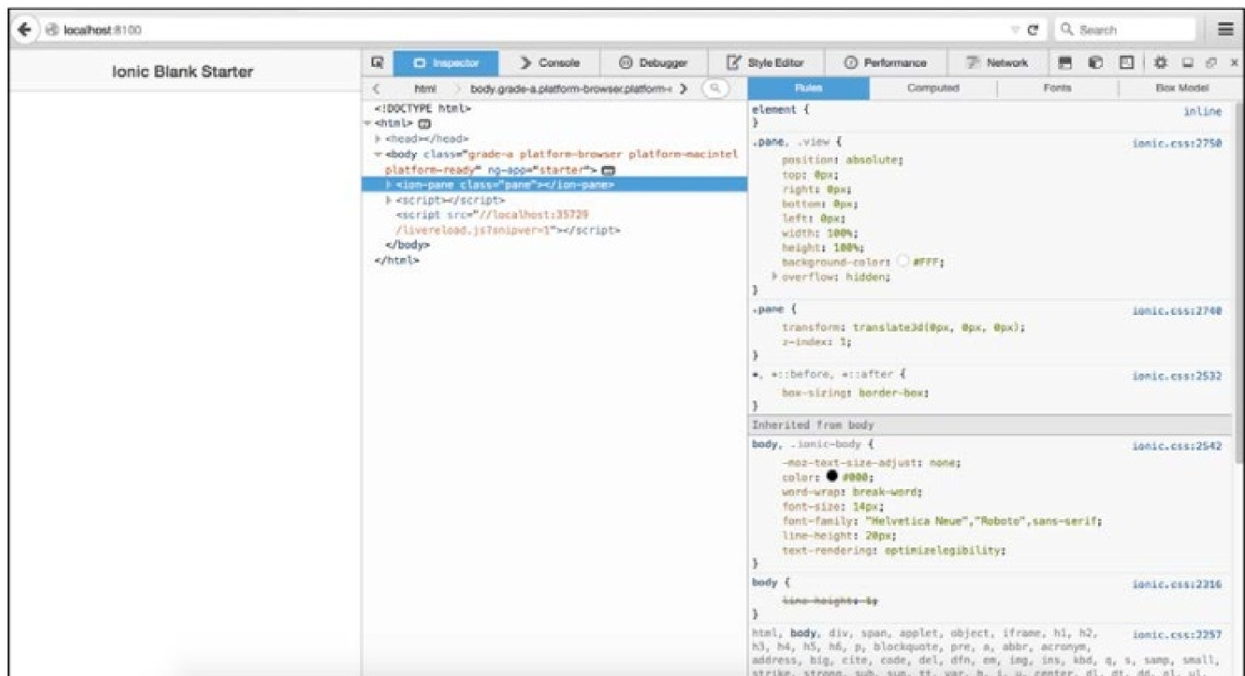
这个视图设置对修复bug和调试非常有用。

Mozilla Firefox

如果你是一个Mozilla Firefox（火狐）粉，上面的效果同样也可以实现。 当Ionic应用启动完成之后，通过 **Cmd + Opt + I**（Mac）和**Ctrl + Shift + I**（Windows/Linux）打开开发者工具（不是Firebug，Firefox的本地开发工具）。 然后点及浏览器窗口顶部按钮，如下：



现在，可以拖动分界条来达成我们在Chrome上达到的效果了。



Ionic项目结构

到目前为止，我们搭建了一个空白Ionic应用，并且在浏览器中启动了。我们现在就过一遍项目结构。

快速提醒你一下，我们知道Ionic是躺在Cordova应用里面的。 所以在我们过一遍Ionic代码之前，我们先来聊聊Cordova包装。

如果已经在你的文本编辑器里面打开了*chapter2 example1*文件夹，你将在项目的根目录下看到如下结构：

```
.
├── bower.json
├── config.xml
├── gulpfile.js
├── hooks
├── ionic.project
├── package.json
├── plugins
├── scss
└── www
```

以下简单解释一下每个项目：

- bower.json：这个文件是由需要通过Bower加载的依赖库组成。后续我们将安装其他的Bower包以在应用中使用。

- config.xml：这个文件是由Cordova在将你的Ionic应用转换成指定平台安装包的所需元信息所组成。如果你打开config.xml， 你将会看到大量的XML标签描述你的项目 。我们将会再次详细阅读此文件。
- gulpfile.js：这个文件是我们在开发Ionic应用期间需要用到的构建任务所组成。
- ionic.project：这个文件是由Ionic应用的相关信息组成。
- hooks：这个文件夹是由Cordova任务执行时候执行的脚本所组成。Cordova任务可以是以下这些： *after_platform_add*（添加了新的平台之后），*after_plugin_add*（添加了新的插件之后），*before_emulate*（模拟之前）， *after_run*（app运行之后），等等。每一个任务都放在一个单独的文件夹，文件夹名字是Cordova任务的名字。当你打开*hooks*文件夹的时候， 你将会看到一个 *after_prepare*和一个*README.md*文件。在*after_prepare*文件夹里面，你会找到一个脚本文件名为*010_add_platform_class.js*。 这个脚本文件将会在Cordova的准备任务执行完成之后被执行。这个脚本做的事情是给标签添加一个class，这个class的名字和应用运行的平台的名字一样。 这样可以帮助我们更好的基于平台为应用进行风格化。你可以在hooks文件夹下的*README.md*文件里找到一个可以勾搭的任务列表。
- plugins：这个文件夹是由所有添加到本项目的插件所组成。我们后续会添加一些其他的插件，然后你就可以在这里看到反应。
- scss：此文件夹是由我们将要用来风格化Ionic组件样式的的*scss*文件所组成。更多相关内容参考第四章 *Ionic与SCSS*
- www：这个文件夹是有Ionic代码所组成。你在此文件夹里面写下的任何东西都将呈现在web view中 。这也是我们花费时间最多的地方。

config.xml文件

*config.xml*文件是一个平台无关的配置文件。如前所述，这个文件是由Cordova在将你的Ionic应用转换成指定平台安装包的所需元信息所组成。
*config.xml*文件的设置是基于W3C的Packaged Web Apps(Widgets)规格书（http://www.w3.org/TR/widgets/）的，扩展到为指定Cordova核心API功能，插件， 和指定平台设置。有两种设置你可以添加到这个文件。一个全局的（global），适用于所有设备；另一个就是指定平台的。
如果在文本编辑器中打开*config.xml*的话，第一个遇到的标签上XML的根标签。接下来，你可以看到widget标签：

```
<widget id="app.example.one" version="0.0.1" xmlns="http://www.w3.org/ns/widgets" xmln
s:cdv="http://cordova.apache.org/ns/1.0">
```

上面指定的*id*是app域名的倒置，这个是我们在搭建项目的时候提供的。其他规格都是定义在widget标签里面作为他的子标签的。 子标签包括app名字（在设备上安装完成之后显示在app图标下面的名字），app描述信息，以及作者详情。
同时，他也有在转换*www*文件夹里面的代码为本地安装器的时候所需要的附加配置。

content标签定义了应用的启动页面。access标签定义了app中允许加载的URL。他默认是加载所有的URL。 preferrence标签里面是一系列的名值对。例如，*DiallowOverScroll*描述的是当用户滚动超过文档顶部或者底部的时候，是否需要视觉反馈。
更多关于平台特殊配置（platform-specific configuration），请参考此链接：

- Android：
  http://docs.phonegap.com/en/4.0.0/guide_platforms_android_config.md.html#Android%20Configuration
- iOS:
  http://docs.phonegap.com/en/4.0.0/guide_platforms_ios_config.md.html#iOS%20Configuration

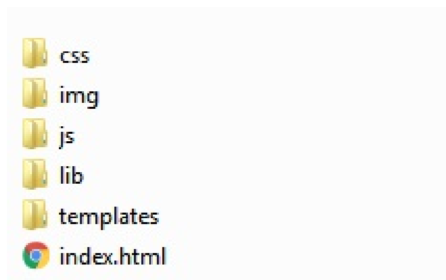
  > 平台特殊配置和全局配置是同样重要的。关于全局配置，请参考：http://docs.phonegap.com/en/4.0.0/config_ref_index.md.html#The%20config.xml%20File

www文件夹

如前所述，这个文件组成了我们的Ionic应用，HTML，CSS以及JS代码。当你打开*www*文件夹，你会看到如下结构：

```
.
├── css
|   └── style.css
├── img
|   └── ionic.png
├── index.html
├── js
|   └── app.js
└── lib
    └── ionic
```

这个图不全



没错，这个图是我本地的

让我们详细的看一下这些东西：

- index.html 这个是应用的启动文件。*config.xml*里面的的*src*标签就指向的这个文件。由于我们使用AngularJS作为我们的JavaScript框架。 此文件用作我们的单页面应用的基本页/主页是再理想不过了。当你打开*index.html*的时候，你将会发现一个ng-app属性，这个属性在js/app.js里面定义指向到了开始模块。
- css：这个文件夹是有咱们app使用的特有样式组成。
- img：这个文件夹里面都是咱们app需要使用的图片。
- js：这个文件夹里面都是咱们app需要使用到的JavaScript代码。我们也是在这里写AngularJS代码的。打开app.js文件，就会发现里面设置好了AngularJS模块，并且传入了Ionic作为依赖。
- lib：这里是我们通过bower安装的包的存放处。当我们创建app的时候，这个文件夹是随之建立的，里面也加载了Ionic文件。 如果你想要重新下载一遍素材与其依赖，可以通过在终端/控制台cd进入到example1文件夹，然后运行以下命令：

  ```
  bower install
  ```

  然后你就可以看到下载了额外的4个文件夹。这些依赖都是在ionic-bower包里面列出的在项目的根目录的bower.json文件里面展示的。
  理想状态下，我们不一定需要在我们的app里面使用这些依赖库。相反，我们更倾向于使用建立在这些依赖库之上的Ionic 包。

这样咱们就看完了这个空白模板。在开始搭建另一个模板之前，我们快速的瞥一眼*www/js/app.js*。
如你所见，我们创建了一个名为i*estarter*的AngularJS模块，然后将*ionic*注入其中。
*$ionicPlatform*服务注入到*run*方法中作为依赖。这个服务用来检测当前工作平台，同时也会处理设备（Android）上的物理按钮。当前内容中，我们使用的是 *$ionicPlatform.ready*方法，这个方法用来在设备准备完成之后进行相关操作。
最好的做法或者说在某些案例中必需要这么做：将你所有的代码包含到*$ionicPlatform.ready*中去。这样一来，你的代码将只会在整个app初始化之后执行。
目前为止，你可能在使用AngularJS进行开发的时候用到的是他的Web方面。但是当使用Ionic的时候，我们还需要使用AngularJS代码里面设备功能相关的代码。 Ionic给我们提供了组织好的了服务来达成这些功能。我们可以回首第一章 Ionic - 搭载AngularJS的自定义服务，并且，我们 后续将会在第五章 Ionic 指令与服务中更加深入Ionic服务。

搭建tabs模板

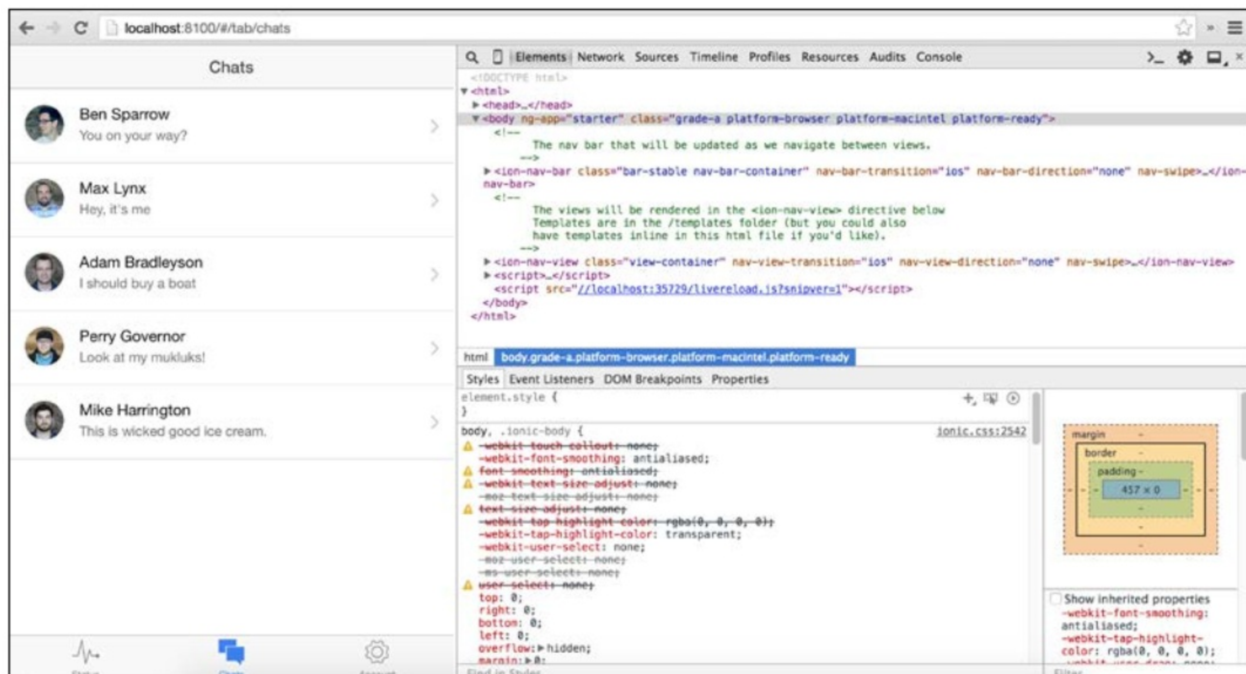
为了对Ionic CLI和项目结构有个更好的感觉，我们也会搭建另外两个开始模板。先来tabs模板。
使用*cd*密码返回*chapter2*文件夹，然后运行以下爱命令：

```
ionic start -a "Example 2" -i app.example.two example2 tabs
```

你可以看到，*example2*文件夹里面就建好了一个新的tabs模板了。使用*cd*命令进入*example2*然后执行以下命令：

```
ionic serve
```

你将会看到如下截屏的标签界面应用：



标签排在页面底部。我们将在第三章 Ionic CSS 组件与导航以及 第五章 Ionic 指令与服务中深入订制方法的知识。

当你返回*example2*文件夹里面分析项目结构的时候，所有的内容基本与空白模板一样，除了*www*文件夹里面的内容。

这一次，你会发现一个新的文件夹叫做*templates*。这个文件夹将由每个AngularJS路由页面部分所组成。*js*文件夹里面也会有两个新的文件：

- controller.js：这里是由AngularJS的控制器代码所组成
- services.js：这里是由AngularJS的服务代码所组成

现在，你大概对 **Ionic**是如何与**AngularJS**整合以及所有的组件是如何手牵手的去进行协作 有了一个比较好的理解。 当我们再深入了解Ionic之后，我们将会对这个结构有更多的感觉。

搭建侧面菜单模板

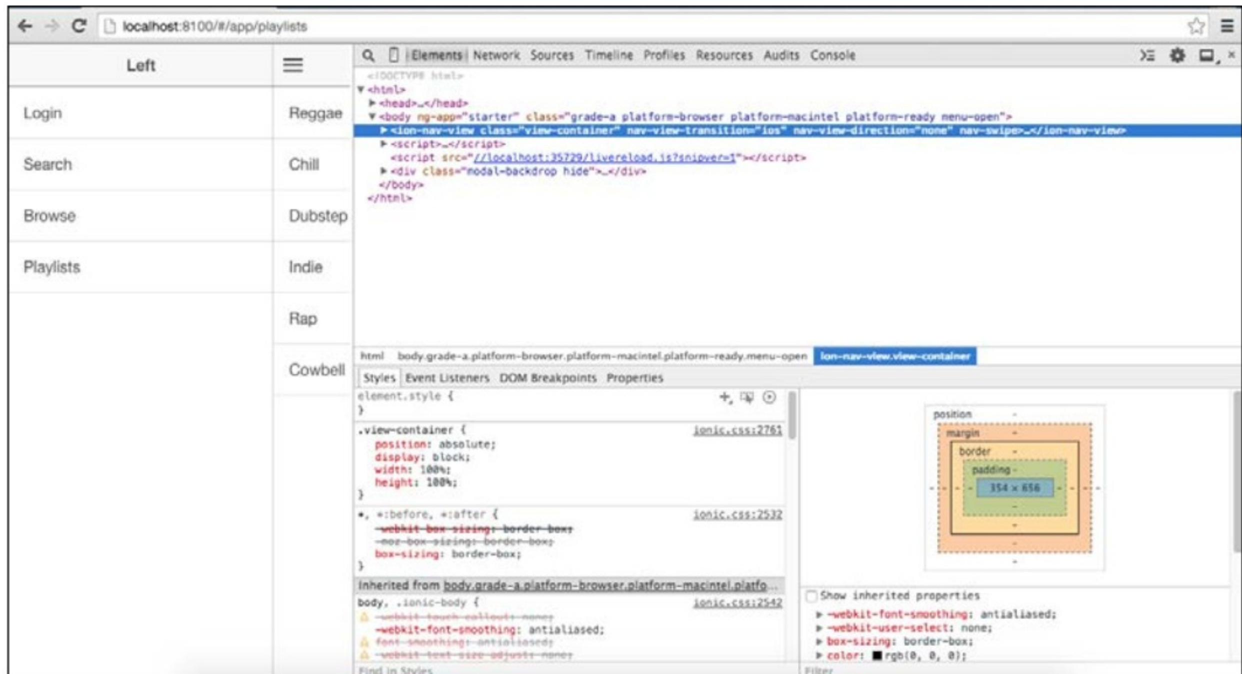
现在我们将进行最后一个模板的搭建。使用*cd*命令回到*chapter2*，然后运行以下命令：

```
ionic start -a "Example 3" -i app.example.three example3 sidemenu
```

然后使用*cd*命令进入*example3*文件夹，运行如下命令：

```
ionic serve
```

然后将会输出类似如下截屏的效果：



你可以自己分析项目结构，然后对比一下较之其他两个模板有何不同。

> 你可以使用*ionic start -l*或者*ionic templates*查看可用的模板列表。然后使用*ionic start*任务试着去搭建这些模板。

generator-ionic

> 用到的名字，有更好的想法请联系我：
>
> - host v. 主导
> - scaffold 搭建，脚手架
> - generator 生成器
> - build 构建
> - confguration-over-code 编码之上配置
> - code-over-confguration 配置之上编码

我觉得虽然Ionic CLI非常不错了，但是他没有相对应的工作流。我说的工作流指的是开发代码和生产代码之间的分界。在通过Ionic CLI搭建的项目中， *www*文件夹主导了开发代码和生产代码。当你的代码越来越多的时候，这会成为一个需要面对的问题了。
这就是generator-ionic价值体现的地方了。generator-ionic是一个用来搭建Ionic项目的Yeoman生成器。如果你还不知道Yeoman是什么东西的话， 那么我就告诉你吧。Yeoman是一个使用Grunt，Bower以及Yo搭建app的脚手架工具。并且，他们很快就会支持gulp了。

> 为什么选择**Yeoman**？ 和其他语言的IDE不同的是JavaScript或者说web开发没有统一的开发环境，IDE里用户只需导航到 **File | New | AngularJS**项目或者**File | New | HTML5**项目。这也是Yeoman为什么适合的地方。

Ionic有自己的CLI来搭建app。但是对于其他没有生成器的框架，Yeoman提供了基本的生成器。

> 想要了解Yeoman更多信息，请参考：http://yeoman.io/，想要研究Yeoman生成器，请参考：http://yeoman.io/generators/

Ionic也有其他一些生成器，但是我对generator-ionic的工作流和功能情有独钟。

安装generator-ionic

安装generator之前，我们需要全局安装*yo*，*grunt*，以及*grunt-cli*。使用如下命令安装即可：

```
npm install yo grunt grunt-cli -g
```

Grunt是另一个与Gulp类似的构建工具。Grunt和Gulp最大的不同是：Grunt是一个编码之上配置的构建工具，而Gulp是配置之上编码的构建工具。

> 更多关于Grunt的信息参考： http://gruntjs.com/ 关于我对Gulp与Grunt的见解，这里有更详细的解读：http://arvindr21.github.io/building-n-Scaffolding

接下来，我们将要全局安装generator-ionic：

```
npm install generator-ionic -g
```

> 安装的时候带有-g标识的值需要安扎ungyici就够了。不用每次使用的时候都去安装一遍。

现在，我们可以使用generator-ionic创建一个新的 Ionic项目了。通过*cd*命令进入到*chapter2*，然后在其中建立一个新的文件夹名为*example4*。然后运行如下命令：

```
yo ionic example4
```

与Ionic CLI不同的是，你需要回到一些关于你想要怎么创建你的应用的问题。你可以参考以下回答：

```
? Would you like to use Sass with Compass (requires Ruby)?
  N
? Which Cordova plugins would you like to include?
  org.apache.cordova.device
  org.apache.cordova.console
  com.ionic.keyboard
? Which starter template would you like to use?
  Tabs
```

Yeoman将会下载项目运行所需的所有的东西。一旦Yeoman搭建完成之后，你可以进入到*example4*文件夹。你会看到里面会有大量的文件以及文件夹。

> 关于完整项目结构，可以参考： https://gitHub.com/diegonetto/generatorionic#project-structure

Ionic-CLI搭建的app和generator-ionic搭建的app有一些很关键的不同点：

- app：与Ionic CLI搭建的app不同的是，我们的开发将在*app*文件夹中进行，而不是*www*文件夹内。这就是我之前提到的代码分界。 我们在*app*文件夹中进行开发，然后运行构建脚本以清理文件和将他们放到*www*文件夹内共生产之用。
- hooks：*hooks*文件夹里面会多出了4个脚本。
- Gruntfile.js：与Ionic CLI不同的是generator-ionic使用Grunt管理任务。如果你觉得这里需要学习的东西太多的话，我建议你用Ionic CLI与GULP，而不是generator和Grunt。

> 如果你使用generator-ionic搭建app的话，不要在动*www*里面的东西。当你运行*build*命令的时候，这个文件夹里面的内容将会被清空，然后从*app*文件夹中重新生成。可以通过这个地址查看运行app时可以用到的工作流命令：https://gitHub.com/diegonetto/generatorionic#workflow-commands 所有的CLI方法都被*grunt*命令包装起来了。 因此，例如当你想要执行Ionic服务的时候，在使用generator-ionic的情况下通过运行*grunt serve*即可。

所以，我们通过以下命令来运行搭建的app吧：

```
grunt serve
```

你可以看到跟用Ionic CLI搭建的tab app一样的输出。
还有三个使用generator-ionic而不是Ionic CLI的理由是他支持以下：

- 代码提醒：https://github.com/diegonetto/generatorionic#grunt-jshint
- 使用Karma（一个测试框架）进行测试，使用Istanbul进行覆盖：https://github.com/diegonetto/generator-ionic#grunt-karma
- Ripple模拟器：https://github.com/diegonetto/generatorionic#grunt-ripple 想你举证使用generator-ionic的主要原因是当你的app变大的时候，你可以导入这个工作流。再次声明，这个个人推荐，可能你喜欢Ionic CLI也说不定。
  你也可以使用其他你觉得用起来比较舒服的Ionic生成器。

总结

本章中，你收获了一些移动混合应用架构的知识。同时你也学会了混合app是如何工作的。我们也看到了app中Cordova是如何将HTML，CSS,以及JS代码缝合到一起然后在web view中执行的。 然后，我们在本地安装了Ionic开发需要的软件。我们使用Ionic CLI搭建了一个空白模板并且分析了他的项目结构。紧接着，我们搭建了另外两个模板并且区分了他们之间的同步。 我们还安装了generator-ionic并且用他搭建了一个范例app，分析了他与Ionic CLI搭建的项目之间的不同点。

> 更多信息可以查看：http://ionicframework.com/present-ionic/slides

接下来的章节我们将理解Ionic CSS组件以及路由。这些知识会帮助我们使用Ionic API搭建有趣的用户界面和多页面应用。

Ionic CSS组件与导航

> 用到的名词，如果有更合适的请联系我：
>
> - grid system 格子系统

到目前为止，我们知道了什么是Ionic，在移动混合应用开发的大舞台上他适合做什么。我们也了解了两种搭建Ionic app的方法：Ionic CLI与generator-ionic。 本章，我们将学习Ionic CSS组件，Ionic格子系统，以及Ionic状态路由。我们将要使用丰富多彩的Ionic组件来创建优秀用户体验的app。

本章将要涵盖的范围：

- Ionic格子系统
- 丰富的CSS组件
- 整合Ionic CSS组件与AngularJS
- Ionic状态路由器
  > 可以通过Github地址获得本章源代码，发起issue，以及与作者沟通，地址是：https://github.com/learning-ionic/Chapter-3

Ionic CSS 组件

Ionic是一个强力的移动CSS框架与一系列牛逼的AngularJS指令与服务的组合。有了这些，任何想法投入市场需要的时间极大的缩小了。 Ionic CSS框架包含了创建一个app所需要的绝大部分组件。
为测试驱动CSS组件的可用性，我们将搭建一个空白的启动版本，然后添加Ionic组件。
开始搭建之前，新建一个文件夹名为*chapter3*，本章的所有范例都将在此文件夹下搭建。

> 本章中，我们每个组件建立一个app以便更好的理解。当然，如果你想要只用一个app也是可以的。

新建一个空白app的命令如下：

```
ionic start -a "Example 5" -i app.example.five example5 blank
```

> 用到的名词，如有更好的建议请联系我
>
> - layout 布局
> - FlexBox 弹性盒
> - grid 格子，栅格
> - class 类
> - markup 标记
> - viewport 视窗，窗口
> - header 头，页头
> - footer 脚，页脚

Ionic 格子系统

你需要使用Ionic提供的格子系统对组件布局进行一个良好的管理。
Ionic格子系统的一个优秀之处是他是基于FlexBox的。FlexBox，或者说CSS Flexible Box Layout Module（CSS弹性盒布局模块）提供了一个盒子模型供用户界面设计。

> FlexBox的更多信息参考： http://www.w3.org/TR/css3-flexBox/，以及一个很不错的FlexBox手册：https://css-tricks.com/snippets/css/a-guide-to-flexbox/

基于FlexBox的格子系统的好处是你不需要任何一个固定行的格子系统。你可以随你乐意在一列中铺任何行数，格子系统将会自动帮你设置到相同宽度。 这一点不同与其他基于CSS的格子系统，你无需担心添加到格子系统中的行的class数量。
为了初步了解格子系统，打开*exmaple5/www*文件夹里面的*index.html*，在*ion-content*里面加上以下代码：

```
<div class="row">
    <div class="col">col-20%-auto</div>
    <div class="col">col-20%-auto</div>
    <div class="col">col-20%-auto</div>
    <div class="col">col-20%-auto</div>
    <div class="col">col-20%-auto</div>
</div>
```

然后在 标签后加上以下样式，以区分不同内容：

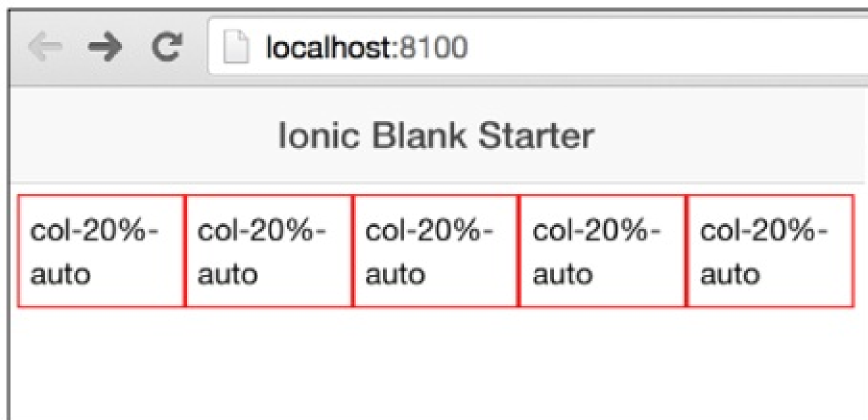
```
<style>
.col {
    border: 1px solid red;
}
</style>
```

> 以上样式不是使用格子系统的必须样式；他在这里只是用来明显区分布局里面列之间的分界线。
>
> 译者测试：使用yo建立的项目， style.css的路径和 app.js的文件路径可能会有问题，这个地方自己改就可以了，有可能是generator-ionc的版本比较旧或者是其他原因造成的

保存文件，然后通过*cd*进入到*example5*文件夹然后运行以下命令：

```
ionic serve
```

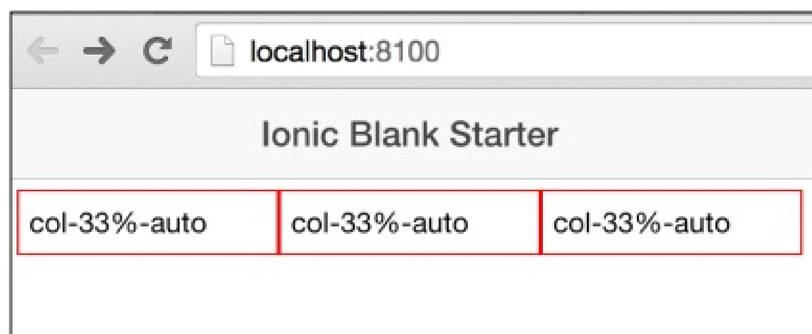
然后你将看到如下效果：



为验证宽度是否是自动变化的，我们将子*div*缩减到3个，如下：

```
<div class="row">
    <div class="col">col-33%-auto</div>
    <div class="col">col-33%-auto</div>
    <div class="col">col-33%-auto</div>
</div>
```

之后你可以看到：

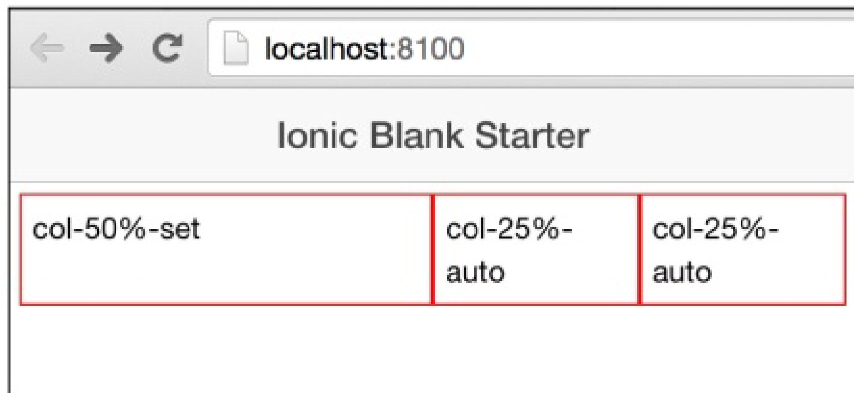


没有任何麻烦，也不需要任何计算；你需要做的仅仅是添加你要使用的列进去就可以了，他们将会自动帮你调整到相等的宽度。
但是这也意味着你不能使用自定义的宽度。想到使用自定义宽度可以简单的通过使用Ionic提

供的类就可以达成。
例如，假设你要将之前例子中的第一列跨度改为50%其余2列使用剩下的宽度；你需要做到的是给第一个*div*添加一个名为*col-50*的类，如下：

```
<div class="row">
    <div class="col col-50">col-50%-set</div>
    <div class="col">col-25%-auto</div>
    <div class="col">col-25%-auto</div>
</div>
```

然后你将看到：



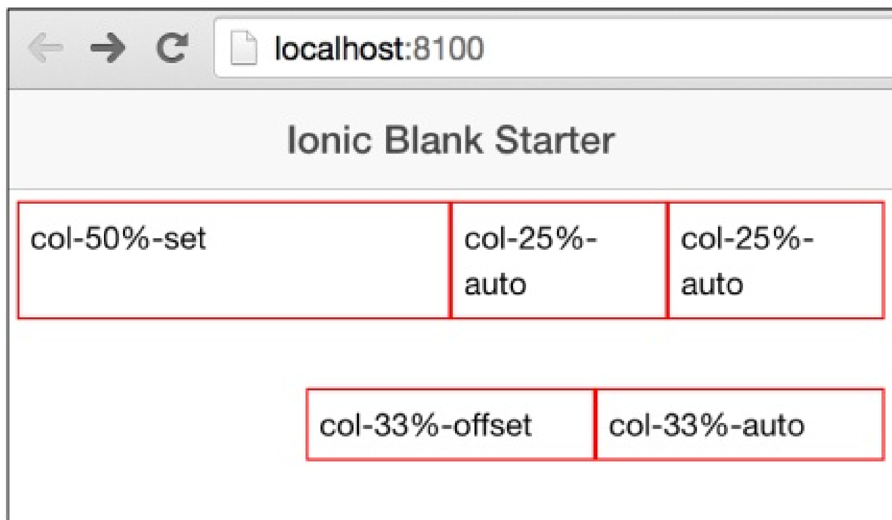
以下列表是关于宽度使用方面的一些预定义的类：

Class name	Percentage width
.col-10	10 percent
.col-20	20 percent
.col-25	25 percent
.col-33	33.3333 percent
.col-50	50 percent
.col-67	66.6666 percent
.col-75	75 percent
.col-80	80 percent
.col-90	90 percent

对于所有的*col*类，你可以使用上表中的任意类来制定宽度。
你也可以对列进行偏移。例如，将以下标记加入到我们的范例中：

```
<div class="row">
    <div class="col col-offset-33">col-33%-offset</div>
    <div class="col">col-25%-auto</div>
</div>
```

然后你将看到：



第一个*div*偏移量33%，其他两个*div*瓜分了剩下的*~66%*。*offset*类做的是在*div*左边进行一个指定比例的填充。

以下是一份应用宽度偏移的预定义类的列表：

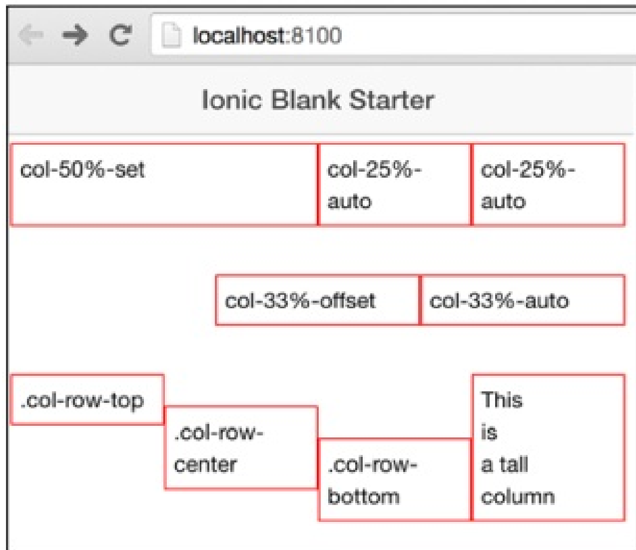
Class name	Percentage width
.col-offset-10	10 percent
.col-offset-20	20 percent
.col-offset-25	25 percent
.col-offset-33	33.3333 percent
.col-offset-50	50 percent
.col-offset-67	66.6666 percent
.col-offset-75	75 percent
.col-offset-80	80 percent
.col-offset-90	90 percent

你也可以在垂直方向上排列这些列。这是格子系统中使用FlexBox带来的另一个好处。

在早先添加了*offset*的列后面添加以下标记：

```
<div class="row">
    <div class="col col-top">.col-top</div>
    <div class="col col-center">.col-center</div>
    <div class="col col-bottom">.col-bottom</div>
    <div class="col">1
        <br>2
        <br>3
        <br>4
    </div>
</div>
```

结果如下：



如果你行中的列有高于其他列的，你可以给那一列添加*col-top*类以将他的内容定位的本行的顶部，如上。 或者你可以添加*col-center*类以将他的内容在本行中居中，或者*col-bottom*以将他的内容相对本行进行底部对齐。
有了这个简单而强大的格子系统，布局是无限可能的。

> 在第六章 创建一个书店*APP*中，我们将会研究响应式格子（布局）和使用*ng-repeat*构建一个动态格子（系统）。 更多关于Ionic格子系统的信息，请参考：http://ionicframework.com/docs/components/#grid

> 用到的名字，如有更好的建议，请联系我：
>
> - item 条目
> - button 按钮
> - class-named 类名为
> - list 列表
> - element 元素
> - header 页头
> - footer 页脚

页面结构

接下来，我们将要了解单页面Ionic应用需要的页面结构。接下来的内容我们可以新建一个空项目（空白模板的项目）。
运行以下命令以创建一个空app：

```
ionic start -a "Example 6" -i app.example.six example6 blank
```

使用*cd*命令，进入*example6*文件夹然后运行以下命令：

```
ionic serve
```

你将会在默认的浏览器中看到空白app启动了。
打开*example6/www/index.html*文件。在*body*标签中，你应该看到这样的一个结构：

```
<ion-pane>
    <ion-header-bar class="bar-stable">
        <h1 class="title">Ionic Blank Starter</h1>
    </ion-header-bar>
    <ion-content>
    </ion-content>
</ion-pane>
```

整个页面都被包装在一个*ion-pane*指令中。
*ion-pane*是一个简单的适配窗口的容器。http://ionicframework.com/docs/api/directive/ionPane/
接下来，你可以看到一个*ion-header-bar*指令(http://ionicframework.com/docs/api/directive/ionHeaderBar/)。这个指令给页面添加了一个

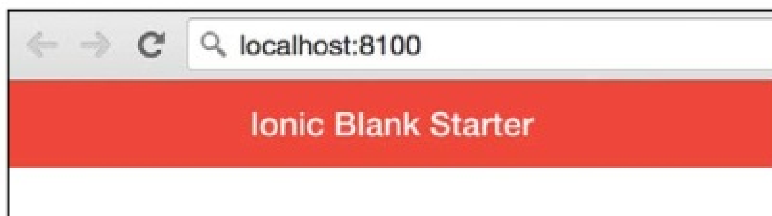
固定的页头。要注意一下添加给*ion-header-bar*指令的类属性。

除此以外，Ionic有9个心情的颜色样本。如下：

light
stable
positive
calm
balanced
energized
assertive
royal
dark

你可以看到我们当前*ion-header-bar*指令应用了*bar-stable*类。可以通过将stable替换为上表中的任意名字来改变页头。

例如，如果将*bar-stable*替换为*bar-assertive*，页头的背景色将会改变，效果看起来将是这样子的：



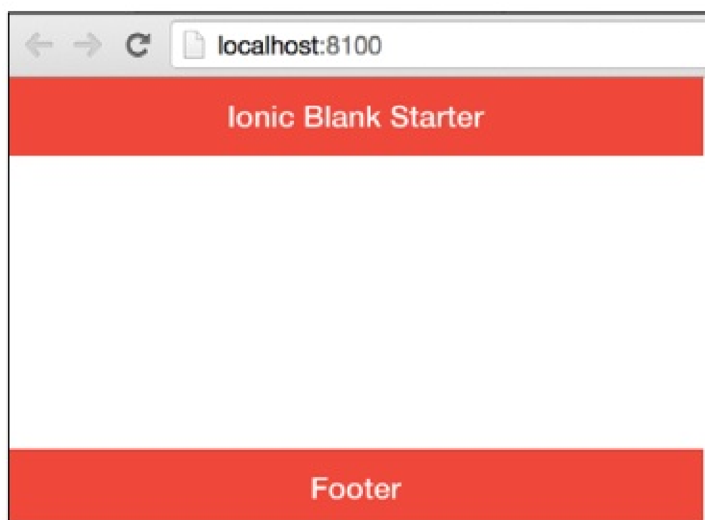
简单快捷，对吧？下一章中，我们将要学习使用SCSS覆盖默认的颜色样本。

页面上*ion-header-bar*后面的指令是*ion-content*。 *ion-content*（http://ionicframework.com/docs/api/directive/ionContent）指令启用了构建内容区域，这篇区域是屏幕的不动产，且要滚动。 同时你也可以通过*$ionicScrollDelegate*更好的对他进行控制。这部分的详细信息，参考 第五章 *Ionic*指令与服务。

为了完善页面结构，我们将会给他添加一个页脚（footer）。在*ion-pane*的末尾，添加：

```
<ion-footer-bar class="bar-assertive">
    <div class="title">Footer</div>
</ion-footer-bar>
```

然后，保存文件；在浏览器中，你将看到如下：



你可以像上面这样使用Ionic搭建一个单页面app，这样子去构建你的app：

```
<body ng-app="starter">
    <ion-pane>
        <ion-header-bar class="bar-assertive">
        ...
        </ion-header-bar>
        <ion-content>
        ...
        </ion-content>
        <ion-footer-bar class="bar-assertive">
        ...
        </ion-footer-bar>
        </ion-pane>
</body>
```

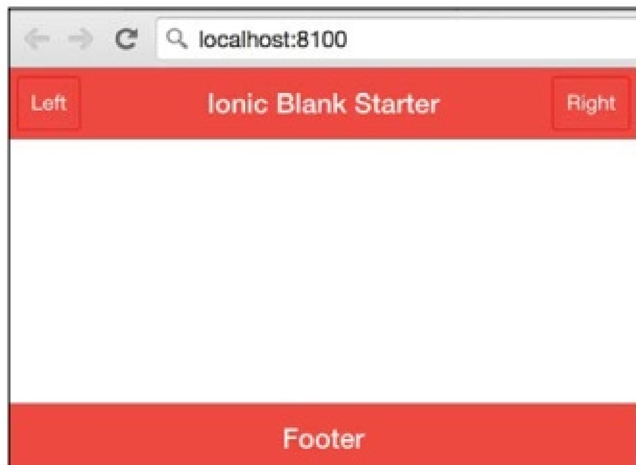
以上结构的无指令版本：

```
<div class="pane">
    <div class="bar bar-header bar-assertive">
    ...
    </div>
    <div class="content has-header has-footer padding">
    ...
    </div>
    <div class="bar bar-footer bar-assertive">
    ...
    </div>
</div>
```

当然，你也可以很方便的给页头或者页脚添加按钮。在header里面添加按钮的结构大概是这样子的：

```
<ion-header-bar class="bar-assertive">
    <div class="buttons">
        <button class="button">Left</button>
    </div>
    <h1 class="title">Ionic Blank Starter</h1>
    <div class="buttons">
        <button class="button">Right</button>
    </div>
</ion-header-bar>
```

*ion-header-bar*中，*h1*指令前添加的按钮将会显示在左边，*h1*标签后面的将显示在页头的右边。如下：



在页脚中使用同样的标注也可以同样生成这些按钮。
*ion-header-bar*指令是生成页头的一个途径。*ion-header-bar*用来生成静态页头是完美之选。但是当我们引入Ionic状态路由的时候，这些事情很快就变得棘手起来。 当你在处理一个多页面应用的时候，你想让Ionic基于导航自动显示返回按钮，我们将会使用*ion-nav-bar*替代*ion-header-bar*。 稍后我们在学习Ionic状态路由的时候会讲到*ion-nav-bar*。

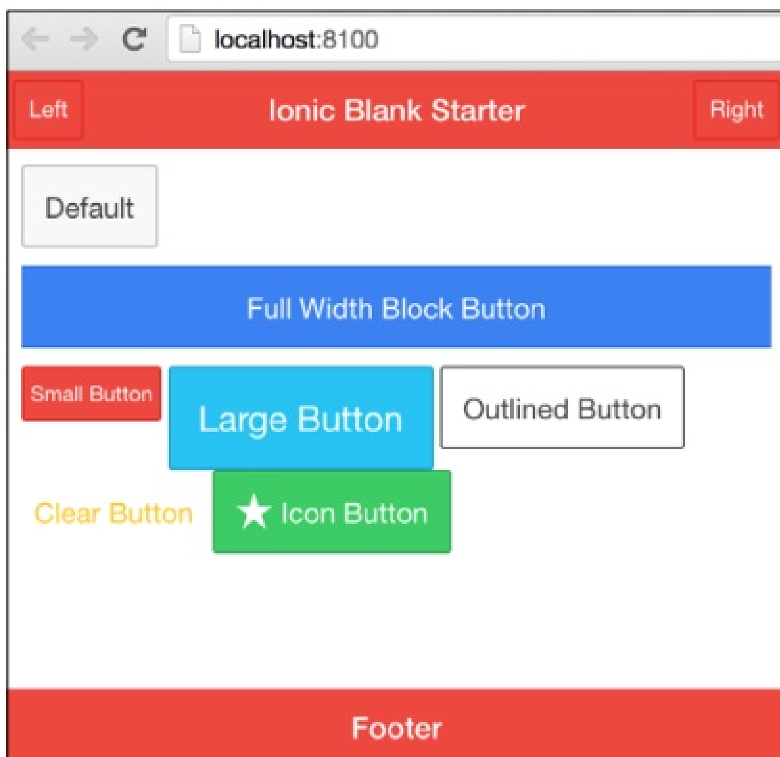
> 更多关于页头组件的信息，请参考：http://ionicframework.com/docs/components/#header。 更多关于content组件信息，请参考： http://ionicframework.com/docs/components/#content 更多关于页脚组件的信息，请参考： http://ionicframework.com/docs/components/#footer

按钮

Ionic在尺寸和样式上，为按钮提供了丰富多彩的变化。
在*www/index.html*的*ion-content*指令中，更新如下代码你就会看到不同按钮的变化:

```
<ion-content class="padding">
    <button class="button">
    Default
    </button>
    <button class="button button-full button-positive">
    Full Width Block Button
    </button>
    <button class="button button-small button-assertive">
    Small Button
    </button>
    <button class="button button-large button-calm">
    Large Button
    </button>
    <button class="button button-outline button-dark">
    Outlined Button
    </button>
    <button class="button button-clear button-energized">
    Clear Button
    </button>
    <button class="button icon-left ion-star button-balanced">
    Icon Button
    </button>
</ion-content>
```

注意看*ion-content*指令的类属性。这个将会给*ion-content*指令的元素间加上一个10px的间隔。保存文件之后，可以看到如下效果：



以上截屏显示了所有基于Ionic颜色样本的按钮。

> 更多关于按钮组件的信息，请参考：http://ionicframework.com/docs/components/#buttons

列表

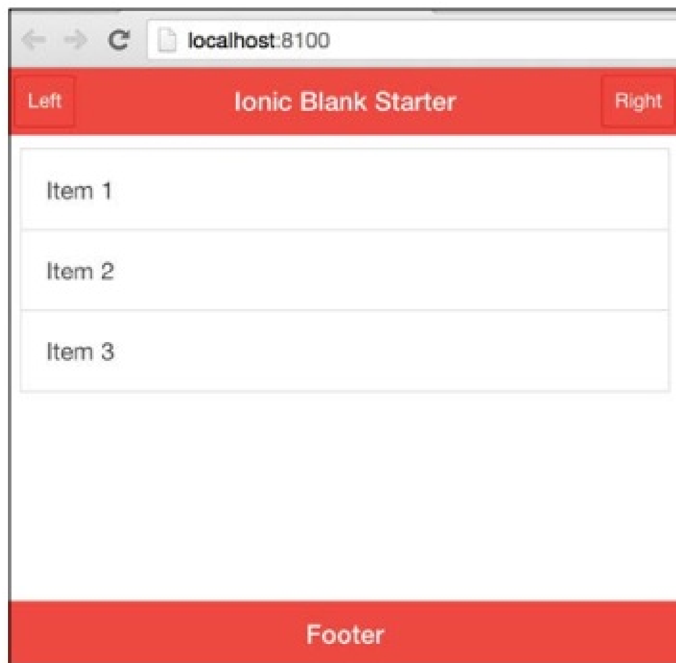
任何app最具代表性的组件莫过于显示一系列条目的列表了。列表和其他Ionic CSS组件一样，结构非常简单，列表都是由CSS类和HTML结构驱动的。在Ionic中，如果你有一个带有名为list的类名父元素和任意数量的带有类名item的子元素，这些条目会以Ionic式列表的方式自动排列。例如：

```
<ul class="list">
    <li class="item">
    Item 1
    </li>
    <li class="item">
    Item 2
    </li>
    <li class="item">
    Item 3
    </li>
</ul>
```

你也可以这么写：

```
<div class="list">
    <div class="item">
    Item 1
    </div>
    <div class="item">
    Item 2
    </div>
    <div class="item">
    Item 3
    </div>
</div>
```

两者都会导向同样的布局显示，如下：



基于我至今为止的Ionic经验，当列表有超过250个通过ng-repeat与对象数组创建的条目的时候，并且每个对象有超过10个属性，这个时候应用的响应就会变慢。 就这么说，你可以根据这个限制基于需求去优化app的执行效率。

Ionic组件的多功能性都在他的类里面。大部分你想要的布局这些类都提供。

例如，如果你想给个在每个列表条目的左边添加图标，你只需要给条目加一个名为*item-icon-left*的类就可以了。这样，条目的左边就会为添加图标留出足够的空间。

> 可以参考这个范例: http://ionicframework.com/docs/components/#item-icons

同样的，当你想要在每个条目的左边添加一个缩略图的时候，你只需要添加一个名为*item-thumbnail-left*的类就可以了。

> 缩略图的范例可以参考： http://ionicframework.com/docs/components/#item-thumbnails
> 关于列表更多信息，请参考： http://ionicframework.com/docs/components/#list

卡片 (Cards)

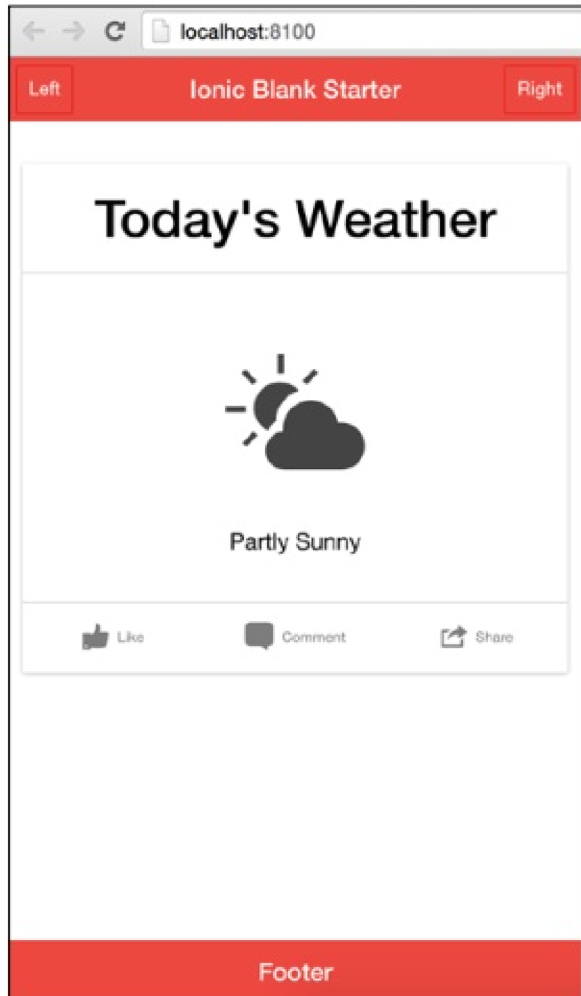
卡片上移动设备上最好的内容陈列设计模式。对于用来展示用户个人内容的页面或者app，卡片上最佳之选。 整个世界在移动设备上展示内容以及桌面上展示某些案例，都在走向卡片模式。 典型的代表有Twitter，和Google Now。

所以，你也可以简单的将这种设计模式带入到你的app中。你需要做的只是将你的个人内容设计为适合卡片展示，然后添加一个名为card的类给容器。 如果你想展示一系列卡片作为列表，你只需要将card类添加到列表容器即可。

以下是一个简单的展示天气信息的卡片秀，如下：

```
<ion-content class="padding">
    <div class="list card">
        <div class="item text-center">
         <h1>Today's Weather</h1>
        </div>
        <div class="item item-body">
        <p>
        <div class="text-center">
        <i class="icon ion-ios-partlysunny"  style="font-size:128px"></i>
        </div>
        <div class="text-center">
        <h2>Partly Sunny</h2>
        </div>
        </p>
    </div>
    <div class="item tabs tabs-secondary tabs-icon-left">
        <a class="tab-item" href="#">
            <i class="icon ion-thumbsup"></i> Like
        </a>
        <a class="tab-item" href="#">
            <i class="icon ion-chatbox"></i> Comment
        </a>
        <a class="tab-item" href="#">
            <i class="icon ion-share"></i> Share
        </a>
        </div>
    </div>
</ion-content>
```

你的页面效果应该是这样的：



这个设计在同时显示了所有的潜在信息的时候又不失优雅。下次你想要向用户展示你的个性的时候，考虑一下使用卡片。

Ionicons 图标

Ionic有大量的字体图标，上面看到的天气图标就是其中一个。导航到http://ionicons.com/，你看到的所有字体图标都可以立即使用。

为了方便起见，提供一个搜索条了来搜索指定类型的图标。例如，当你在搜索条里输入*sunny*，你就可以看到上面截屏里面显示的图标。
重要警告，一定要确保你的Ionicons版本和你的Ionic CSS文件里面的Inicons版本一致。 Ionic团队持续添加新的图标和更新版本号。此处使用的sunny图标是在Ionicons 2.0.1版本发布的。

> 更多关于Ionicons的信息，请参考：http://ionicframework.com/docs/components/#icons

表单元素

Ionic也带来了他的表单元素和布局。 以下是一个简单的登录表单结构：

```
<ion-content class="padding">
    <div class="list">
        <label class="item item-input">
          <span class="input-label">Username</span>
          <input type="text">
        </label>
        <label class="item item-input">
          <span class="input-label">Password</span>
          <input type="password">
        </label>
    </div>
</ion-content>
```

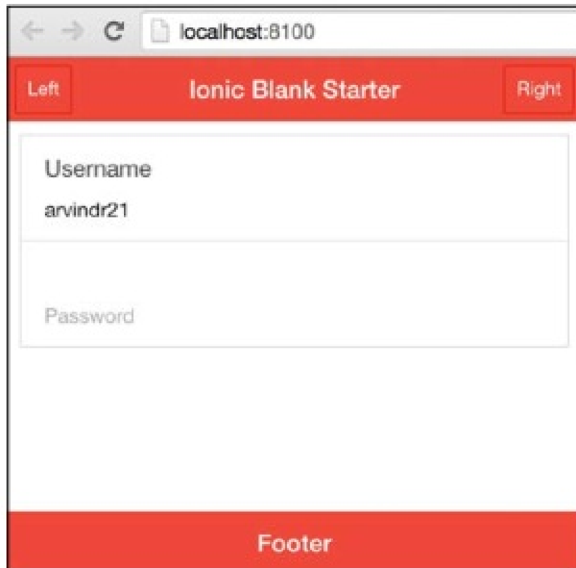
效果图：



你也可以通过给标签添加一个*item-floating-label*类来创建一个花式表单：

```
<ion-content class="padding">
    <div class="list">
        <label class="item item-input item-floating-label">
          <span class="input-label">Username</span>
          <input type="text" placeholder="Username">
        </label>
        <label class="item item-input item-floating-label">
          <span class="input-label">Password</span>
          <input type="password" placeholder="Password">
        </label>
    </div>
</ion-content>
```

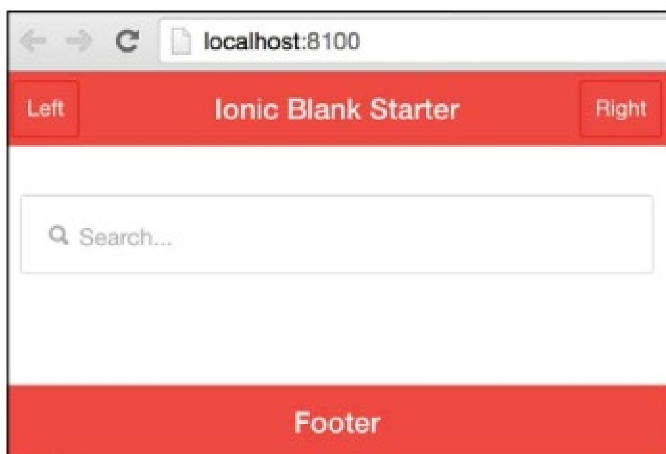
效果图如下：



你也可以给这些表单元素添加图标。只需要在标签（label）里面添加一个带*placeholder-icon*的*i*标签（tag）就可以了：

```
<ion-content class="padding">
    <div class="list list-inset">
        <label class="item item-input">
          <i class="icon ion-search placeholder-icon"></i>
          <input type="text" placeholder="Search...">
        </label>
    </div>
</ion-content>
```

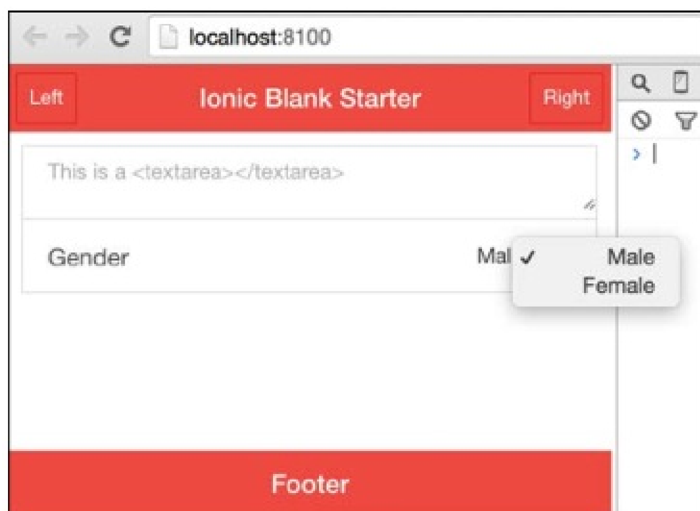
效果图如下：



你也可以添加其他的表单元素，例如文本域(text ares)和选择列表(select)；他们会如期出现并且会非常整齐的融入到其他表单组件中:

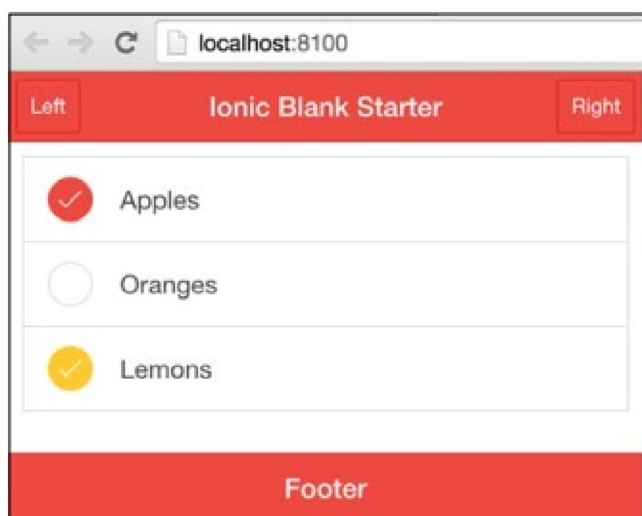
```
<ion-content class="padding">
    <div class="list">
        <label class="item item-input">
          <textarea placeholder="This is a &lt;textarea&gt;
          &lt;/textarea&gt;"></textarea>
        </label>
        <label class="item item-input item-select">
          <div class="input-label">
          Gender
          </div>
          <select>
            <option>Male</option>
            <option>Female</option>
          </select>
        </label>
    </div>
</ion-content>
```

效果图如下：



有两种方法展示复选框。你可以将他展示为复选框，也可以作为切换开关。
以下标记代码展示了一个可选的水果列表：

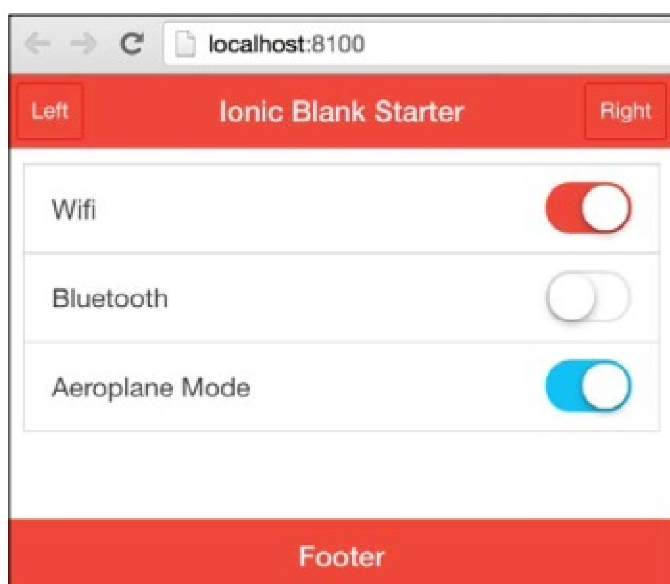
```
<ion-content class="padding">
    <ul class="list">
        <li class="item item-checkbox">
            <label class="checkbox checkbox-assertive">
              <input type="checkbox">
            </label> Apples
        </li>
        <li class="item item-checkbox">
            <label class="checkbox">
              <input type="checkbox">
            </label> Oranges
        </li>
        <li class="item item-checkbox checkbox-energized">
            <label class="checkbox">
              <input type="checkbox">
            </label> Lemons
        </li>
    </ul>
</ion-content>
```



> Ionic给app默认的是iOS样式的主题；因此，你可以看到那个圆圈复选框。在第五章，Ionic指令与服务中我们将学习如果修改他。

以下标记显示了可开闭的的切换开关：

```
<ion-content class="padding">
    <ul class="list">
        <li class="item item-toggle">
            Wifi
            <label class="toggle toggle-assertive">
            <input type="checkbox">
            <div class="track">
              <div class="handle"></div>
            </div>
            </label>
        </li>
        <li class="item item-toggle">
            Bluetooth
            <label class="toggle toggle-positive">
            <input type="checkbox">
            <div class="track">
              <div class="handle"></div>
            </div>
            </label>
        </li>
        <li class="item item-toggle">
            Aeroplane Mode
            <label class="toggle toggle-calm">
            <input type="checkbox">
            <div class="track">
              <div class="handle"></div>
            </div>
            </label>
        </li>
    </ul>
</ion-content>
```



最后，我们将会给基于Ionic CSS的组件添加一个输入范围。 在处理用户输入的时候，这是一个非常便利和强大的组件。最佳展示范例是一个亮度调节滑块，看起来是这样子的：



以上效果图需要的代码是这样子的：

```
<ion-content class="padding">
    <div class="list">
        <div class="item range range-positive">
            <i class="icon ion-ios-sunny-outline"></i>
            <input type="range" name="volume" min="0" max="100" value="33">
            <i class="icon ion-ios-sunny"></i>
        </div>
    </div>
</ion-content>
```

> 用到的名词：

整合Ionic CSS组件与AngularJS

如果你有一些漂亮的页面，里面有很些很酷的组件，但是他们在实时环境中什么都不做，在这种情况下，我们需要的是什么？ 所以在这个子标题中，我们将要看一下整合这些美丽的Ionic组件与AngularJS以使得我们的页面更加功能化。
第一个例子用来处理的是，在表单域全部有效之前禁用表单提交。我们将要创建一个由邮件地址和密码组成的表单。 在用户输入的邮件和密码的长度最少为3之前，我们将禁用登录按钮。
我们先通过以下命令搭建一个空白模板的的app：

```
ionic start -a "Example 7" -i app.example.seven example7 blank
```

接下里，我们将要对*index.html*进行更改，给他添加一个表单和一个名为*ng-disabled*的按钮。当邮件的模型值和密码的模型值都是*false*的时候*ng-disabled*的值为*true*。

> 关于JavaScript中眞(truthy)与假(falsy)的概念，请参考：
> http://adripofjavascript.com/blog/drips/truthyand-falsy-values-in-javascript.html

*www/index.html*文件里面相关代码应该是这样的：

```
<div class="list">
    <label class="item item-input">
      <span class="input-label">Email</span>
      <input type="email" ng-model="email">
    </label>
    <label class="item item-input">
      <span class="input-label">Password</span>
      <input type="password" ng-model="password" ng-minlength="3">
    </label>
    <div class="padding">
        <button ng-disabled="!email || !password" class="button button-block button-po
sitive">Sign In</button>
    </div>
</div>
```

保存文件，然后运行以下命令：

```
ionic serve
```

如果表单里面没有输入，或者输入了无效的数据，按钮将被禁用，大概是这样子的：

如果表单输入有效，按钮将会被激活：

这个简单的示例展示了AngularJS和Ionic如何一起工作来创建一个伟大的用户体验的。 上面的范例可以扩展显示有效信息。

接下来的范例中，我们将处理一个稍微复杂一些的Ionic和AngularJS的整合。我们将实现一个简单的打分小部件。 这个小部件是由5个镂空的小星星组成的。当用户点击其中任何一个小星星来打分的时候，我们将会对从开始的那个星星到用户点击的那个星星进行填充。

运行以下指令以创建一个新的空白模板项目：

```
ionic start -a "Example 8" -i app.example.eight example8 blank
```

接下来，在*www/js/app.js*中添加以下控制器代码：

```
.controller('MainCtrl', ['$scope', function($scope) {
    $scope.ratingArr = [{
        value: 1,
        icon: 'ion-ios-star-outline'
    }, {
        value: 2,
        icon: 'ion-ios-star-outline'
    }, {
        value: 3,
        icon: 'ion-ios-star-outline'
    }, {
        value: 4,
        icon: 'ion-ios-star-outline'
    }, {
        value: 5,
        icon: 'ion-ios-star-outline'
    }];
    $scope.setRating = function(val) {
        var rtgs = $scope.ratingArr;
        for (var i = 0; i < rtgs.length; i++) {
            if (i < val) {
                rtgs[i].icon = 'ion-ios-star';
            } else {
                rtgs[i].icon = 'ion-ios-star-outline';
            }
        };
    }
}])
```

如上代码所示，我们创建了一个名为*ratingArr*的数组。这个数组元素由两部分组成，星星的值和需要应用到星星上的样式。 我们也创建了另一个名为*setRating()*的方法，这个方法将在星星被点击的时候进行调用。这个方法读取作为参数传入的星星的值。 然后，我们遍历从开始的星星到选中的星星的所有的打分对象的，我们将图标设置为填满的星星，其他的作为轮廓。

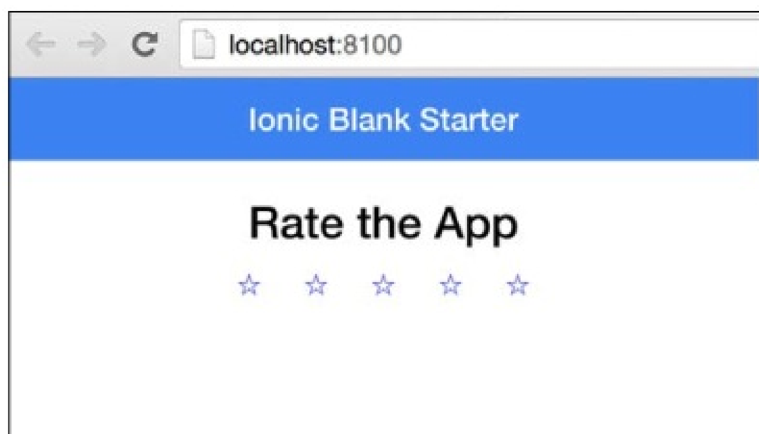
*www/index.html*的body部分将是如下：

```
<body ng-app="starter" ng-controller="MainCtrl">
    <ion-pane>
        <ion-header-bar class="bar-positive">
            <h1 class="title">Ionic Blank Starter</h1>
        </ion-header-bar>
        <ion-content class="padding">
            <div class="padding text-center">
                <h3>Rate the App</h3>
                <div>
                    <a href="javascript:" ng-repeat="r in  ratingArr" class="padding"
style="text-decoration:none;">
                        <i class="icon {{r.icon}}" ng-click="setRating(r.value)"></i>
                    </a>
                </div>
            </div>
        </ion-content>
    </ion-pane>
</body>
```

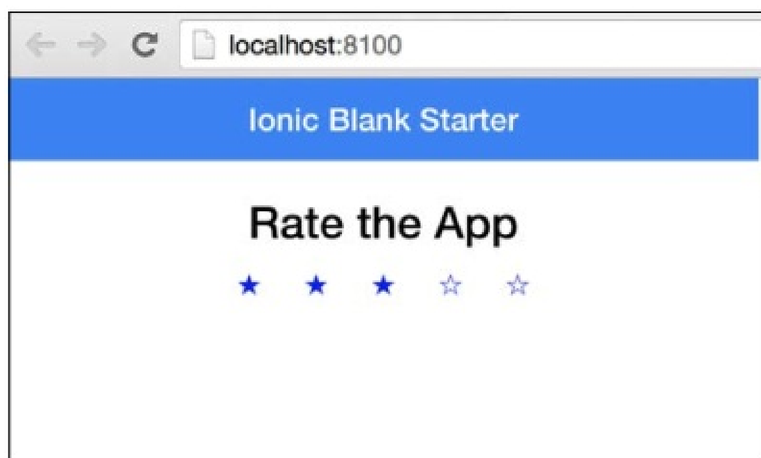
我们给*body*标签添加了*ng-controller*指令，在*ion-content*指令里面，我们有添加了*div*用来在遍历*ratingArr*的时候渲染星星。
保存完所有文件后，运行:

```
ionic serve
```

你可以看到如下：



当你选择了第三个星星的时候，显示效果差不多是这样子的：



完成这些之后，我们已经简单理解了如何整合Ionic CSS组件和AngularJS。下一个主题中，我们将学习AngularUI路由器。

> 用到的名词：
>
> - router n. 路由器
> - route v. 路由
> - AngularUI Router AngularUI路由
> - padding 填充
> - named views 命名视图

Ionic路由器

在应用比较小只有几个页面的情况下，维护状态和管理数据将会比较简单。但是当应用变得越来越复杂的时候，处理模板，模板数据路由相关数据等等，将会变的很难。

因此，为了使管理复杂多页面Ionic应用变得简单，我们使用Ionic路由器。Ionic路由器和AngularUI路由器一样。 更多信息请参考： https://github.com/angular-ui/ui-router

在AngularUI路由器文件里：

> AngularUI Router is a routing framework for AngularJS, which allows you to organize the parts of your interface into a state machine. Unlike the $route service in the Angular ngRoute module, which is organized around URL routes, UI-Router is organized around states, which may optionally have routes, as well as other behavior, attached.
> AngularUI路由器是AngularJS的路由框架，他允许将你的接口组织到一个状态机里面。与AngularJS ngRouter里面的$routeservice以URL路由组织不同的是，UI-Router是围绕状态来组织路由的，它可以选择性地拥有路由，行为及附件。
>
> 更多关于AngularUI 路由的信息，参考：https://github.com/angular-ui/ui-router/wiki

一个简单的双页app

Ionic源码里面捆绑了AngularUI。所以，在我们将Ionic添加为依赖之后，我们可以直接注入*$stateProvider*和*$urlRouterProvider*到我们的*config*方法中来，以此建立路由。 我们将通过一些范例学习路由。
第一个范例中，我们将创建一个双页应用。一个导航按钮在两个页面之中进行导航。 这个范例的目的是了解路由器的语法和设置，这样我们可以将相同的逻辑应用到其他范例中。
我们将创建一个空白模板项目，然后给他添加路由，这样他就变成了一个多页面应用。
执行以下命令创建一个空白模板项目：

```
ionic start -a "Example 9" -i app.example.nine example9 blank
```

一旦app创建成功以后，打开*www/js/app.js*。我们将要创建一个名为*config*的方法然后为我们的app添加路由。 我们将在*www/js/app.js*的*run*方法后面添加以下*config*方法：

```
.config(function ($stateProvider, $urlRouterProvider) {
$stateProvider
    .state('view1', {
        url: '/view1',
        template: '<div class="padding"><h2>View 1</h2><button class="button button-po
sitive" ui-sref="view2">To View 2</button></div>'
    })
    .state('view2', {
        url: '/view2',
        template: '<div class="padding"><h2>View 2</h2><button class="button button-as
sertive" ui-sref="view1">To View 1</button></div>'
    })
    $urlRouterProvider.otherwise('/view1');
})
```

就像你看到的一样，*$stateProvider*和*$urlRouterProvider*作为依赖注入到了*config*方法。 参考主页可知这些服务是和Ionic包一起发布出来的。
接下来，我们使用*$stateProvider*来定义应用的状态。在这个案例中，状态和视图是一样的。*$stateProver*上的*state*方法是用来声明路由的。方法的第一个参数是状态的可读名。第二个参数是由路由配置组成的一个对象。 作为路由配置的一部分，我们提供了一个URL和当触发URL时用做渲染的一个模板。
在上面的配置中，我们创建了两个状态：一个叫做*view1*，这个将在导航到*http://localhost:8100/#/view1*的时候激活， 第二个叫做*view2*，这个将在导航到*http://localhost:8100/#/view2*的时候激活。
当你观察URL的时候，会发现在view的名字的前面有一个#(hash 哈希)。这个哈希告诉浏览器不需要向服务器发起资源请求；取而代之的是，这些资源都是在客户端的， JavaScript框架将会负责去渲染他们。
简单来说，让URL哈希后面的任何东西改变的时候，会发出一个*hashchange*的事件。路由器有监听器，这些监听器会在事件发出的时候被激活。 这个监听器将负责根据哈希值（*view1*或者 *view2*）和他的状态配置来管理UI。（简单讲，当哈希改变的时候，路由将为改变了的哈希调用对应的控制器和模板）
注意，我们已经为视图编写了渲染用的模板。在下一个范例中我们将学习使用外部文件作为模板。 同时按钮有一个名为*ui-sref*的指令（http://angular-ui.github.io/uirouter/site/#/api/ui.router.state.directive:ui-sref）。 *ui-sref*指令用于将链接绑定到状态。如果状态有一个关联的URL，这条指令将自动生成和更新*href*。
所以，在我们的场景中，当我们点击*view1*模板里面呈现的那个按钮的时候，app将导航到*view2*，反之亦然。
最后，我们给*config*方法提供一个默认的URL作为结束：

```
$urlRouterProvider.otherwise('/view1');
```

在上面这一行代码中，我们指定了默认的URL，当当前URL不能匹配任何配置的状态URL的时候，用户将会被重定向到*view1*状态。/
有了这些，我们就已经成功的设置了状态。但是对于设置来讲，我们还有一个关键的部分。我们需要告知路由器页面的哪些部分需要使用状态的内容去更新。 这一步通过在我们的*index.html*中添加*ion-nav-view*就可以达到了。

> 对于状态路由器来将，ion-nav-view和ui-view同样适用。ion-nav-view扩展自ui-view,然后添加了一些功能例如动画和历史。

在*index.html*中，将*ion-content*替换为以下代码：

```
<ion-nav-view class="has-header"></ion-nav-view>
```

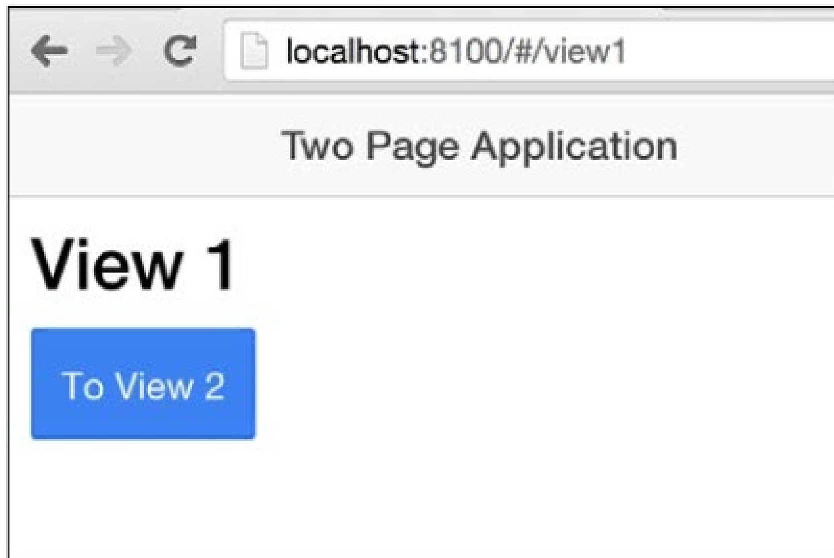
*has-header*类在容器的顶部添加了一个padding。这样就确保了模板内容不会是从页头条的后面开始。
完整的*index.html*的*body*代码如下：

```
<body ng-app="starter">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Two Page Application</h1>
        </ion-header-bar>
        <ion-nav-view class="has-header"></ion-nav-view>
    </ion-pane>
</body>
```

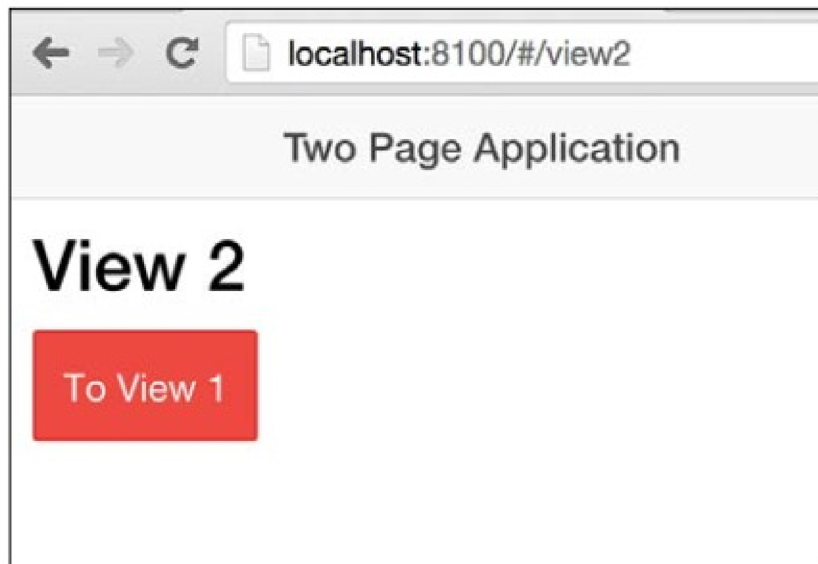
保存所有文件然后运行以下命令：

```
ionic serve
```

你将会看到下面截屏的效果:



当点击 **To View2**按钮的时候，他会将你带到**View2**，如下：



注意观察导航后的URL。
接下来的范例中，我们将在单独的HTML文件中创建模板然后在咱们的路由器中进行配置。同时，我们也将为*config*对象引入新的属性，名为controller。
我们将要创建的是一个双页的app；第一个页面上我们早先创建的登录表单，第二页是那个打分页。
这个范例的目标是理解外部模板和给视图绑定控制器。老规矩，新建一个空白模板项目：

```
ionic start -a "Example 10" -i app.example.ten example10 blank
```

接下来，我们将要设置路由了。给*www/js/app.js*添加一个*config*方法，如下：

```
.config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
        .state('login', {
            url: '/login',
            templateUrl: 'templates/login.html',
            controller: 'LoginCtrl'
        })
        .state('app', {
            url: '/app',
            templateUrl: 'templates/app.html',
            controller: 'AppCtrl'
        })
    $urlRouterProvider.otherwise('/login');
})
```

我们有两个状态:*login*和*app*。我们用来一个新的属性名为*templateUrl*替换之前的*template*。*templateUrl*是模板文件的位置。模板文件可以是硬盘上的一个单独的文件，也可以是*index.html*的一部分，例如*script*标签。两种途径我们会都试验一下。
我们也添加了一个名为*controller*的新属性。这个属性告诉路由器当导航到一个路由的时候需要调用哪一个控制器。如你所见，我们将要为两个控制器创建两个视图。
为了使用基于*script*标签的模板开始，我们将创建两个空的控制器。在*www/js/app.js*文件的*run*方法后面，加上以下代码：

```
.controller('LoginCtrl', function ($scope) {
})
.controller('AppCtrl', function ($scope) {
})
```

由于我们已经在*route*配置中声明了控制器，AngularJS会在导航到视图的时候去查找相应的控制器。 有鉴于此，我们创建了两个假的控制器。功能代码稍后添加进去。
接下来，在我们的*www/index.html*中，我们将使用以下代码替换其中的*ion-content*：

```
<ion-nav-view class="has-header"></ion-nav-view>
```

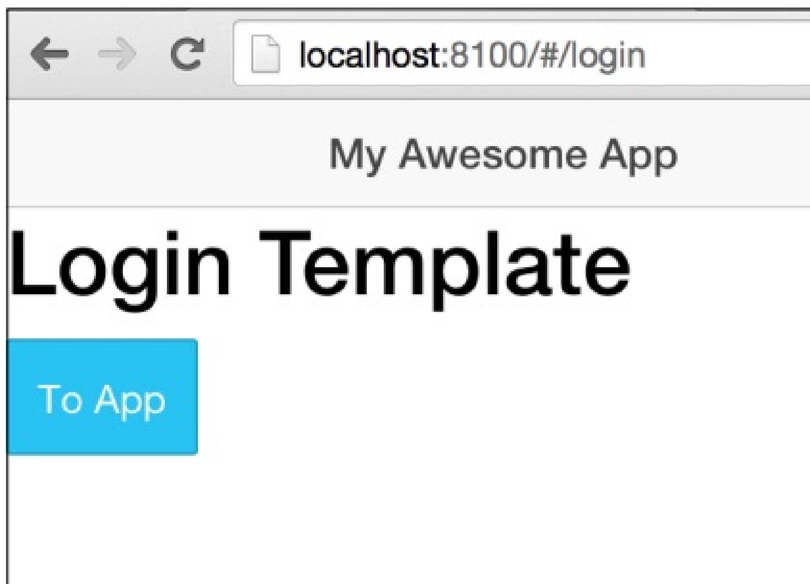
你可以在*body*标签的任何地方添加这个模板。*www/index.html*的*body*部分代码如下：

```
<body ng-app="starter">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">My Awesome App</h1>
        </ion-header-bar>
        <ion-nav-view class="has-header"></ion-nav-view>
    </ion-pane>
    <script type="text/ng-template" id="templates/login.html">
        <h1>Login Template</h1>
        <button class="button button-calm" ui-sref="app">To App</button>
    </script>
    <script type="text/ng-template" id="templates/app.html">
        <h1>App Template</h1>
        <button class="button button-royal" ui-sref="login">To
        Login</button>
    </script>
</body>
```

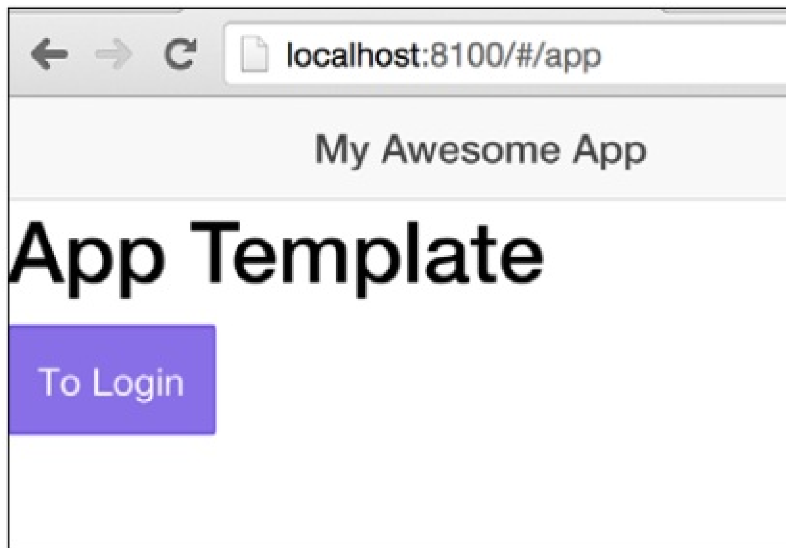
留意*script*标签上的*id*属性。他们跟*templateUrl*是一样的。这就是用来连接脚本*tag/ng-template*到路由*templateUrl*的钩子。
保存所有文件，运行如下命令：

```
ionic serve
```

你将看到如下结果:



当点击**To App**按钮的时候，将会看到如下视图：



上面的范例示范了如何使用*script*标签在HTML文件里写模板的。
在继续我们的真实案例之前，移除我们在*www/index.html*里面的内联模板。
我们将在*www*文件夹里面创建一个名为*templates*的文件夹-不是项目根目录，是*www*文件夹，千万记住了。
在*templates*文件夹里面，新建一个文件名为*login.html*。文件内容和我们在*example7*中用的是一模一样的。*login.html*文件看起来是酱紫的：

```
<div class="list">
    <label class="item item-input">
      <span class="input-label">Email</span>
      <input type="email" ng-model="email">
    </label>
    <label class="item item-input">
      <span class="input-label">Password</span>
      <input type="password" ng-model="password"
      ng-minlength="3">
    </label>
    <div class="padding">
        <button ui-sref="app" ng-disabled="!email || !password" class="button button-b
lock button-positive">Sign In</button>
    </div>
</div>
```

需要注意的是，我们给按钮添加了*ui-sref*。在按钮没有被激活之前，点击这个按钮是不会被重定向到app视图的。
接下来，在*templates*文件夹下面新建一个名为*app.html*的文件。文件内容和我们在*example8*里面的是一样的，如下：

```
<div class="padding text-center">
    <h3>Rate the App</h3>
    <div>
        <a href="javascript:" ng-repeat="r in ratingArr" class="padding" style="text-d
ecoration:none;">
            <i class="icon {{r.icon}}" ng-click="setRating(r.value)"></i>
        </a>
    </div>
    <button ui-sref="login" class="button button-block
    button-clam">Sign Out</button>
</div>
```

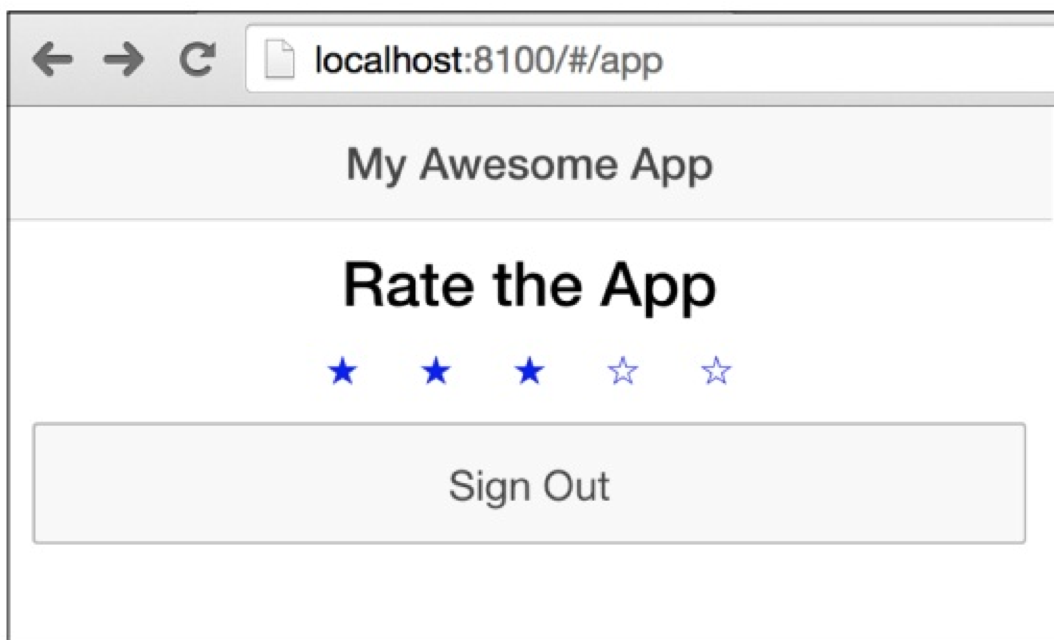
当你保存好了这些文件之后，回到页面，你就会发现login页面出现了。当你输入了一个有效的邮件地址和一个超过3个字符的密码的时候，sign-in按钮将被激活。 当你点击**Sign in**按钮的时候，他会把你带到app视图去。

> 如果你没看到更新后的UI，这意味着你还没有删除*www/index.html*里面的基于脚本的模板。*scirpt*标签写入的模板的优先级高于硬盘里面的模板， 且硬盘里面的模板需要通过AJAX请求才能得到。关于更多的AngularJS模板缓存，请参考：https://docs.angularjs.org/api/ng/service/$templateCache

打开app视图的时候，会发现小星星部件了。这是因为我们的小星星都是基于一个名为*ratingArr*的区域变量的。我们将会像在*example8*里面那样更新我们的*AppCtrl*:

```
.controller('AppCtrl', function($scope) {
    $scope.ratingArr = [{
        value: 1,
        icon: 'ion-ios-star-outline'
    }, {
        value: 2,
        icon: 'ion-ios-star-outline'
    }, {
        value: 3,
        icon: 'ion-ios-star-outline'
    }, {
        value: 4,
        icon: 'ion-ios-star-outline'
    }, {
        value: 5,
        icon: 'ion-ios-star-outline'
    }];
    $scope.setRating = function(val) {
        var rtgs = $scope.ratingArr;
        for (var i = 0; i < rtgs.length; i++) {
            if (i < val) {
                rtgs[i].icon = 'ion-ios-star';
            } else {
                rtgs[i].icon = 'ion-ios-star-outline';
            }
        };
    }
})
```

现在，当你返回查看视图的时候，会发现小星星又出现了，当你点击他们的时候，一切如你预期一样：



虽然，我们将*LoginCtrl*留空。只要你想要，你就可以给**Submit**按钮绑定一个*ng-click*然后在控制器里面调用一个方法做你的有效验证。 你可以从按钮标签移除*ui-sref*属性使用控制器里面的*$state*服务导航到app视图。*www/templates/login.html*可以用以下代码替换掉：

```
<button ng-click="validate()" ng-disabled="!email || !password" class="button button-block button-positive">Sign In</button>
```

然后*www/js/app.js*里面的*LoginCtrl*完整代码如下：

```
.controller('LoginCtrl', function($scope, $state) {
$scope.validate = function() {
// some other validations...
$state.go('app');
}
})
```

下一个范例中，我们将使用状态路由建立一个稍微复杂一些的UI。我们将建立一个标签组件。但是，首先，我们的看一下AngularUI路由器里面的命名视图。
假设在这么一个情景里，当我们路由改变的时候页面有3个地方需要更新。使用AngularJS的*ngRoute*的话，是做不到的，因为*ngRoute*路由器只允许我们每个app里面存在一个*ng-view*。但是AngularUI路由器提供了一些名为“命名视图”的东西，在这里你可以在页面上有多个*ui-view*并且对他们命名。 这样，根据视图状态的不同，将会在这些视图里面加载不同的模板。
考虑以下HTML，我们在一个页面上有3个视图部分：

```
<body>
  <div ui-view="partialview1"></div>
  <div ui-view="partialview2"></div>
  <div ui-view="partialview3"></div>
</body>
```

这样，当配置我们的路由的时候，我们将在我们路由配置对象中引入一个新的属性，名为*views*。 随后，我们将设计当状态改变的时候，需要调用哪一个控制器和模板。例如：

```
$stateProvider
    .state('page1', {
        views: {
            'partialview1': {
                templateUrl: 'page1-partialview1.html',
                controller: 'Page1Partialview1Ctrl'
            },
            'partialview2': {
                templateUrl: 'page1-partialview2.html',
                controller: 'Page1Partialview2Ctrl'
            },
            'partialview3': {
                templateUrl: 'page1-partialview3.html',
                controller: 'Page1Partialview3Ctrl'
            }
        }
    })
    .state('page2', {
        views: {
            'partialview1': {
                templateUrl: 'page2-partialview1.html',
                controller: 'Page2Partialview1Ctrl'
            },
            'partialview2': {
                templateUrl: 'page2-partialview2.html',
                controller: 'Page2Partialview2Ctrl'
            },
            'partialview3': {
                templateUrl: 'page2-partialview3.html',
                controller: 'Page2Partialview3Ctrl'
            }
        }
    })
```

如你所见，当你在*page1*状态的时候，三个命名视图都将调用对于的视图和控制器，*page2*也是如此。

为将相同的理念带入Ionic，我们将对*ion-nav-view*指令添加*name*属性然后使用他来进行命名视图的工作。 我们将使用两个或者或者两个标签页来构建一个页面。

还是用以下命令搭建一个新的空白模板项目：

```
ionic start -a "Example 11" -i app.example.eleven example11 blank
```

我们的标签界面将会有两个标签页*-login*和*register*。在模组初始化完成之后，我们将会在*www/js/app.js*中去配置这两个状态：

```
.config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
        .state('login', {
            url: '/login',
            views: {
                login: {
                    templateUrl: 'templates/login.html'
                }
            }
        })
        .state('register', {
            url: '/register',
            views: {
                register: {
                    templateUrl: 'templates/register.html'
                }
            }
        })
    $urlRouterProvider.otherwise('/login');
})
```

注意观察*view*属性，以及他的子属性(*login*和*register*)的名称。这就是我们声明*views*对象的方法。

接下来，我们将会使用Ionic tabs指令(http://ionicframework.com/docs/api/directive/ionTabs/)进行工作。 这是个非常简单的指令，他使用*ion-tabs*指令包装了*ion-tab*指令来创建一个标签界面。

这样一来，在我们的*www/index.html*中，我们将*body*标签部分的代码替换为以下：

```
<body ng-app="starter">
    <ion-nav-bar class="bar-royal">
    </ion-nav-bar>
    <ion-tabs class="tabs-royal">
        <ion-tab icon="ion-power" ui-sref="login">
            <ion-nav-view name="login"></ion-nav-view>
        </ion-tab>
        <ion-tab icon="ion-person-add" ui-sref="register">
            <ion-nav-view name="register"></ion-nav-view>
        </ion-tab>
    </ion-tabs>
</body>
```

如你所见，*ion-tabs*指令是由两个*ion-tab*指令组成的，*ion-tab*指令是由*ion-nav-view*作为内容视图的。 每个*ion-nav-view*都设置了需要加载的*name*属性。

现在我们要做的是搭建这两个模板。在*www*文件夹内新建一个名为*templates*的文件夹，然后在其中新建一个名为*login.html*的文件。 *www/templates/login.html*文件的内容如下：

```
<ion-view view-title="Login">
    <ion-content class="padding">
        <div class="list">
            <label class="item item-input">
              <span class="input-label">Email</span>
              <input type="email" ng-model="email">
            </label>
            <label class="item item-input">
              <span class="input-label">Password</span>
              <input type="password" ng-model="password"
              ng-minlength="3">
            </label>
            <div class="padding">
                <button ng-disabled="!email || !password" class="button button-block b
utton-royal">Sign In</button>
            </div>
        </div>
    </ion-content>
</ion-view>
```

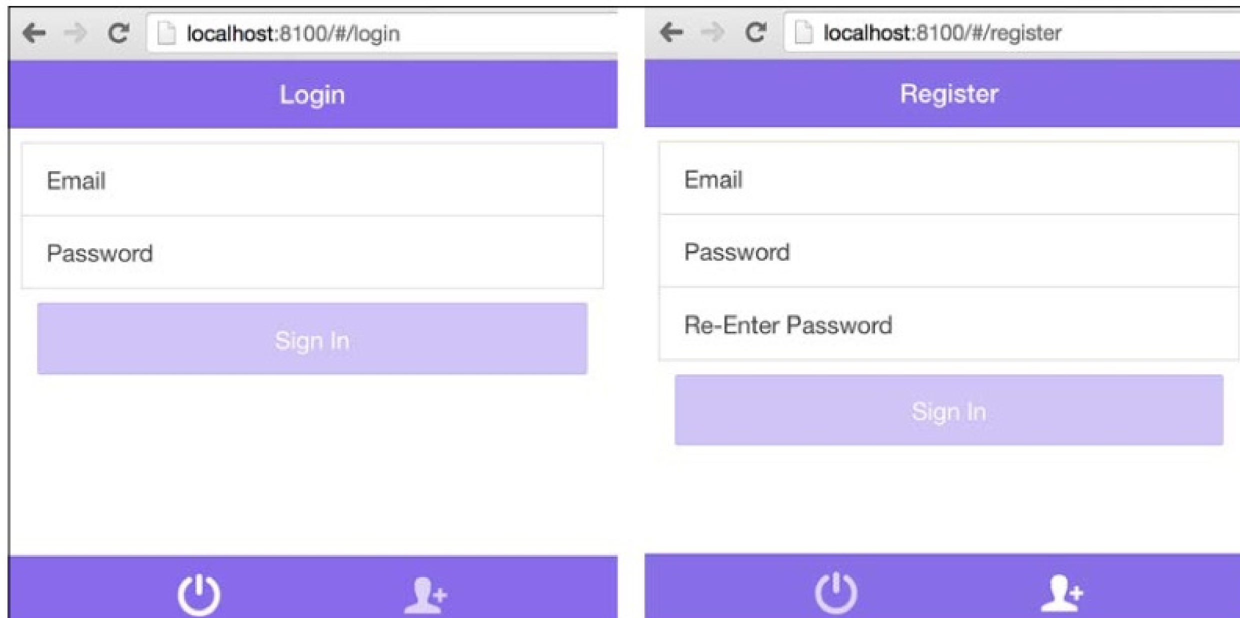
注意看我们是如何在*ion-view*指令里面包装模块的，然后是在*ion-conent*指令里。接下来，在*www/templates*下面， 新建一个名为*register.html*的文件，内容如下：

```
<ion-view view-title="Register">
    <ion-content class="padding">
        <div class="list">
            <label class="item item-input">
              <span class="input-label">Email</span>
              <input type="email" ng-model="email">
            </label>
            <label class="item item-input">
              <span class="input-label">Password</span>
              <input type="password" ng-model="password"
              ng-minlength="3">
            </label>
            <label class="item item-input">
              <span class="input-label">Re-Enter Password</span>
              <input type="password" ng-model="password2"
              ng-minlength="3">
            </label>
            <div class="padding">
                <button ng-disabled="(!email || !password) || (password != password2)"
 class="button button-block buttonroyal">Sign In</button>
            </div>
        </div>
    </ion-content>
</ion-view>
```

保存所有文件然后运行如下命令：

```
ionic serve
```

效果图如下：



在上面的范例中，我们使用开始模板，从无到有的建立了一个标签组件。我们不用每次都这么做。因为我们有对应的项目模板。 我们可以很快的通过以下命令来新建一个标签页项目：

```
ionic start -a "Example 12" -i app.example.twelve example12 tabs
```

一旦标签页模板项目搭建完成，进入*www/js/app.js*然后滚动*config*方法，你将会看到app的路由配置。如下：

```
.config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
        .state('tab', {
            url: "/tab",
            abstract: true,
            templateUrl: "templates/tabs.html"
        })
        .state('tab.dash', {
            url: '/dash',
            views: {
                'tab-dash': {
                    templateUrl: 'templates/tab-dash.html',
                    controller: 'DashCtrl'
                }
            }
        })
        .state('tab.chats', {
            url: '/chats',
            views: {
                'tab-chats': {
                    templateUrl: 'templates/tab-chats.html',
                    controller: 'ChatsCtrl'
                }
            }
        })
        .state('tab.chat-detail', {
            url: '/chats/:chatId',
            views: {
                'tab-chats': {
                    templateUrl: 'templates/chat-detail.html',
                    controller: 'ChatDetailCtrl'
                }
            }
        })
        .state('tab.account', {
            url: '/account',
            views: {
                'tab-account': {
                    templateUrl: 'templates/tab-account.html',
                    controller: 'AccountCtrl'
                }
            }
        });
    $urlRouterProvider.otherwise('/tab/dash');
});
```

如果你有留意的话，会发现*tab*状态属性有一个新的属性叫做*abstract*设置为*true*。抽象状态是不能切换过去的。 他只会在他的一个子类激活的时候被激活。
在我们的场景中，*tab*持有者将会是一个抽象状态，并且当他的子tab激活的时候，这个tab状态都将自动激活。

> 更多关于抽象状态的信息，参考: https://github.com/angular-ui/ui-router/wiki/Nested-States-%26-NestedViews#abstract-states

当你打开*templates/tabs.html*的时候，你可以看到*ion-tabs*指令是设置在一个模板文件里面的而不是像之前的范例一样设置在*index.html*里面的。 这个模板将作为*tabs*组件的抽象状态进行工作。 同时，你也可以发现，*tab-dash.html*，*tab-chats.html*以及*tab-account.html*和我们上一个范例的架构方式是一样的。
使用如下口令运行app进行测试：

```
ionic serve
```

你大概会看到，当你点击*chat*列表里面的一个条目的时候，会把你带到一个新的视图展示详细信息。 这种设置叫做主详情视图（master detail view），“master”是聊天列表，“detail”是聊天详情。 同时，注意看不同的聊天URL不一样，例如： http://localhost:8100/#/tab/chats/0和 http://localhost:8100/#/tab/chats/1 等等。 去*www/js/app.js*中查看*tab.chat-detail*的状态配置的时候，可以看到如下代码：

```
.state('tab.chat-detail', {
    url: '/chats/:chatId',
    views: {
        'tab-chats': {
            templateUrl: 'templates/chat-detail.html',
            controller: 'ChatDetailCtrl'
        }
    }
})
```

*url*属性的值是*'/chats/:chatId'*。注意*chatId*前面的冒号。这里告诉路由器*chatId*是一个动态值；当遇到这个路由的时候，在*chats*部分之后在验证路由，然后将URL里*chats*部分后面的值的存储到一个名为*chatId*的变量里。 现在，当我们实时处理我们的应用的时候，这个值将可以在*$stateParams*上得到。参考*www/js/controllers.js –ChatDetailCtrl*：

```
.controller('ChatDetailCtrl', function($scope, $stateParams, Chats) {
$scope.chat = Chats.get($stateParams.chatId);
})
```

以上代码向你展示如何对标签视图和一个主详情视图进行混合与匹配。
你也可以试试搭建一个*sidemenu*模板然后看侧边菜单是如何配置路由的。

总结

在本章中，我们学习了大部分的Ionic CSS 组件。我们也看过了可用的一些颜色样本。接下来，我们整合Ionic CSS组件与AngularJS并添加了一些功能。 我们从0开始使用Ionic状态路由器进行工作，创建了一个简单的双页面app。最后，我们探索了标签页界面和主详情视图。在接下来的章节里，我们将会学习使用强力的SCSS来定制Ionic CSS。

第四章 Ionic与SCSS

在本章中，我们将学习如何全方位的对Ionic app进行定制。Ionic默认有7种预定义的色调，或者说颜色样本。 在本章中，我们将对这些颜色进行编辑，然后更改Ionic组件的外观和感觉。本章的目的是理解如何使用Ionic SCSS，所以不会专注于特定组件的讲解。

本章中，我们将涵盖以下主题：

- SASS vs. SCSS
- 设置SCSS
- 使用SCSS变量
- 使用SCSS混合式
- 给一个侧边菜单app定制主题

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通: https://github.com/learning-ionic/Chapter-4

> 用到的名词：
>
> - pre-process 预处理器
> - programmable 可编程的
> - semi-colons 分号
> - brace 花括号
> - function 函数
> - mixins 混合式
> - CSS-like synax 类CSS语法

什么是Sass?

根据Sass文档 http://sass-lang.com/documentation/ :

> Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well organized, and get small stylesheets up and running quickly. Sass是一个CSS的扩展，给CSS这门基本语言添加了力量与优雅。他允许你使用变量，嵌入规则，混入，内联导入等等， 所有这些特性都是完全兼容CSS语法的。Sass帮助良好的组织大型的样式表，对于小型表单使它结构简单，运行加速。

简单来说，Sass使得SCSS可编程。你也许会问为什么我们这章讲的是SCSS，而这里却在讲Sass。 是这样的Sass和SCSS一样是一个CSS预处理器，CSS预处理器可以用他自己的方法来编写pre-CSS语法。

Sass是作为另一个名为HAML的预处理器的一部分由一群Ruby开发者开发出来的。 因此，他从Ruby那里继承了大量的语法样式，例如躲进，没有花括号，没有分号等等。
以下是一个Sass文件的范例：

```
// app.sass
brand-primary= blue
.container
    color= !brand-primary
    margin= 0px auto
    padding= 20px

=border-radius(!radius)
    -webkit-border-radius= !radius
    -moz-border-radius= !radius
    border-radius= !radius
*
    +border-radius(0px)
```

当通过一个Sass编译器运行以上Sass代码的时候，他返回了一个老式的良好的CSS。生成的CSS大概是这样的：

```
.container {
    color: blue;
    margin: 0px auto;
    padding: 20px;
}
* {
    -webkit-border-radius: 0px;
    -moz-border-radius: 0px;
    border-radius: 0px;
}
```

但是你有没有注意到在Sass代码中有一个*brand-primary*作为一个变量，在*container*中替换了他的值的， 或者*border-radius*作为一个函数（也可以叫如混合式）当使用参数调用的时候生成CSS规则？这些在CSS都不见了。
哪些使用*bracket-based*编程语言的人们这么些代码有点费劲。于是，SCSS就出现了。
Sass是Syntactically Aswsome Style Sheet（语法牛逼的样式表）的简写， SCSS是Sassy CSS（时髦的CSS）的简写。 SCSS和Sass非常像，除了类CSS语法。如果使用SCSS写入上Sass代码的话，是这样子的：

```
// app.scss
$brand-primary: blue;
.container{
    color: !brand-primary;
    margin: 0px auto;
    padding: 20px;
}
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    border-radius: $radius;
}
* {
    @include border-radius(5px);
}
```

这个看起来离CSS靠近了些，对吧？这个令人印象深刻。Ionic使用SCSS给他的组件样式化。

> 关于更多的SCSS vs. Sass的信息，可以参考这个帖子：
> http://thesassway.com/editorial/sass-vs-scss-which-syntax-is-better

现在，我们对SCSS和Sass有了一个初步的了解，我们将在Ionic app中对组件制定主题的时候对他们进行评估。

> 用到的名词：
>
> - dependencies 依赖库

在Ionic项目中使用SCSS

下面我们将要在已有的Ionic项目中设置SCSS。我们将从新建一个空白模板的项目开始。新建一个文件夹名为*chapter4*然后在其中打开终端/命令行，运行：

```
ionic start -a "Example 13" -i app.example.thirteen example13 tabs
```

我们可以通过两种方式在项目中设置SCSS：

- 手动设置
- Ionic CLI任务

手动设置SCSS

遵循以下步骤，手动设置SCSS：

1. 使用*cd*命令，进入*example13*： **cd example13**
2. 安装所必需的依赖库。利用模板搭建的Ionic项目都会有一个*package.json*文件。这个文件里面有设置SCSS需要的所有的依赖。 同时，项目也有一个*gulpFile.js*文件，这个文件也定义好了SCSS任务，这个任务是用来监视SCSS文件的更改，并即时编译成CSS文件的。
3. 运行以下命令可以安装这些依赖： **npm install**
4. 如果你没有全局安装Gulp，可以通过以下命令全局安装： **npm install gulp --global**
5. 接下来，打开*www/index.html*文件，在*head*标签里面有一行注释掉的代码，大概是这样的：

   ```
   <!-- IF using Sass (run gulp sass first), then uncomment below and remove the CSS
   includes above
   <link href="css/ionic.app.css" rel="stylesheet">
   -->
   ```

   移除掉注释部分，流行*link*标签。接下来，移除掉之前提及的*ionic.css*的引用。因为我们不再需要他了。
6. 回到终端/命令行，运行以下代码: **gulp sass** 这个将会在*www/css*文件夹内生成*ionic.app.css*与*ionic.app.min.css*。

这些是你在Ionic项目里面设置SCSS所需的全部了。稍后我们会学习自定义SCSS。

使用**Ionic CLI**任务设置**SCSS**

现在我们要学习的是使用Ionic CLI设置任务对项目设置SCSS。鉴于我们已经在*example13*里面设置好了SCSS，我们需要新建另一个项目的来学习：

ionic start -a "Example 14" -i app.example.fourteen example14 tabs

接下来，进入*example14*目录，运行如下命令： **ionic setup scss**

这个命令将会下载依赖库，移除*index.html*里面的注释内容，在*www/css*文件夹里面新建*ionic.app.css*和*ionic.app.min.css*文件。这个文件里面有设置SCSS需要的所有的依赖。 就问你，这个屌不屌！

> 用到的名词
>
> - mixin 混合式，混合体

使用Ionic SCSS

这部分涵盖了如何自定义Ionic SCSS变量和混合式。
我们将要编写的代码是基于你已经达到了使用SCSS的基本需求。

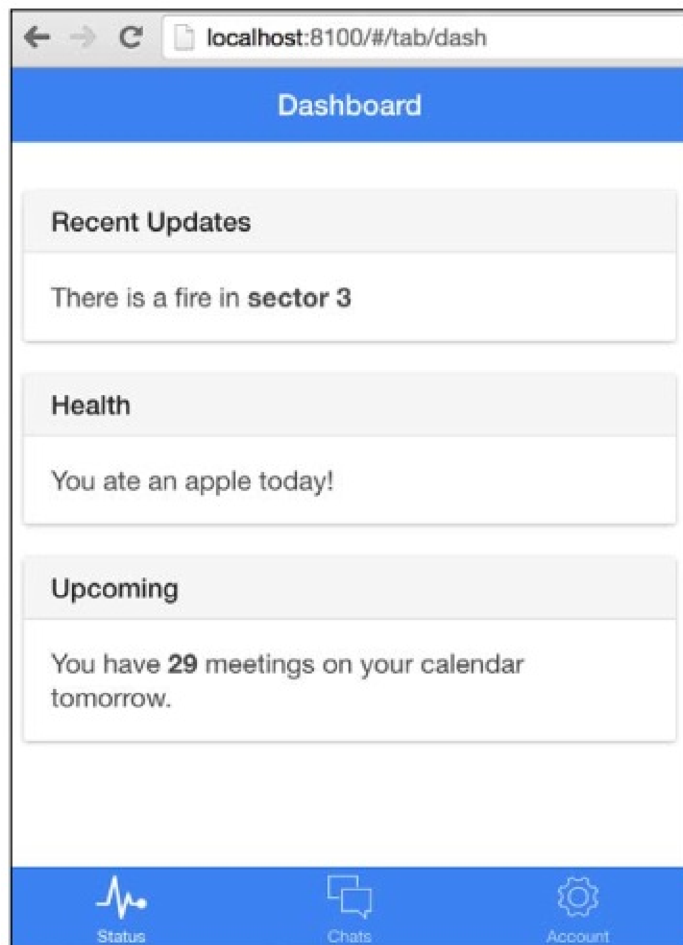
> 如果对于SCSS你是个新手，建议你参考以下指引：http://sass-lang.com/guide

基础样本

早先的时候，我们看过了Ionic提供的一些基本的样色样本：Positive，Assertive,Calm等等。
他们都是Ionic团队预定义和设置好的。 如果你想将组件的颜色改为positive类，怎么办呢？下面就来学习一下。
回到*example14*文件夹，打开*www/index.html*将*ion-nav-bar*指令上的*bar-stable*改为*bar-positive*。 接着，打开*www/templates/tabs.html*文件，移除*ion-tabs*指令上的*tabs-color-active-positive*类，添加*tabs-positive*。

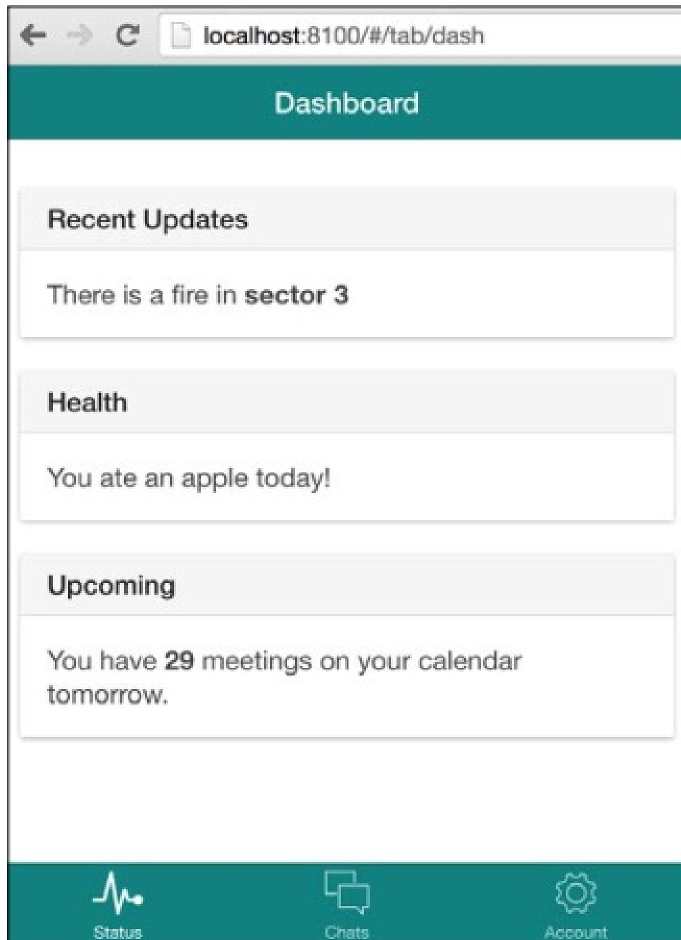
> 编写本书的时候，*tabs*模板对*ion-nav-bar*使用了stable样式

为了直观显示，运行： **ionic serve**
标签界面显示效果如下：



我们假设这是我们最后的app吧。接下来我们将要为这个布局定制主题，这个主题非常简单。所有蓝色将被替换成鸭绿色（teal）。
打开*scss/ionic.app.scss*，复制文件顶部的这行注释： **$positive: #387ef5 !default;**
然后在注释块的后面加上他。接下来，我们将*$positive*变量的值的设为*teal*： **$positive: teal;**

保存文件，然后后台会运行Sass任务，自动生成新的*ionic.app.css*和*ionic.app.min.css*文件。之后页面就自动刷新了。 然后你就可以看到这个鸭绿色主题的页面了：



注意看positive类的所有引用是如何更新到teal的。是不是很屌？管理你的移动app的主题不再像载人航空科技那么复杂了！

理解Ionic SCSS的设置

在这个部分，我们将学习Ionic SCSS是如何建立的。
当你查看项目结构的时候，你会发现一个*scss*文件夹，里面是一个*ionic.app.scss*文件。想要对默认的Ionic主题进行自定义的话，你需要重写此处的变量，且只能是此处的变量。 如果你计划制定多个主题，我建议你在*scss*文件夹里面使用*theme1.scss*，*theme2.scss*这样的文件。
千万记得，所有的主题文件必须有以下这两行：

```
// The path for our ionicons font files, relative to the built CSS in www/css
$ionicons-font-path: "../lib/ionic/fonts" !default;
// Include all of Ionic
@import "www/lib/ionic/scss/ionic";
```

在一个典型的主题文件中，第一部分重写SCSS变量。第二部分加载Ionic核心SCSS文件；最后我们重写生成的类。

Ionic核心SCSS文件引用自以下两个表达式：

```
$ionicons-font-path: "../lib/ionic/fonts" !default;
@import "www/lib/ionic/scss/ionic";
```

如果你之前用过SCSS的话，你应该会知道任何使用*!default*指定的变量在没有指定值之前是没有默认值的，就跟之前范例一样。

为了更好的理解这其中发生了什么，我们先导航到*$ionicons-font-path*变量的路径去，也就是*www/lib/ionic/fonts*。 这个文件夹包含了4个字体文件，这4个字体文件根据浏览器的兼容性而决定使用哪一个。

接着，我们导航至Ionic SCSS框架的所在路径。也就是*www/lib/ionic/scss/ionic*。你可能也发现了，*scss*文件夹内没有叫做*ionic*的文件夹。 因为他是引用自*www/lib/ionic/scss/*文件夹内的*ionic.scss*的。同时注意*scss*文件夹内的其他SCSS文件的名字都是以下划线开头的。

当你打开*ionic.scss*的文件的时候，你会发现，这个文件所做的只是把当前文件夹内的其他的SCSS文件导入进来：

```
@charset "UTF-8";
@import
    // Ionicons
    "ionicons/ionicons.scss",
    // Variables
    "mixins",
    "variables",
    // Base
    "reset",
    "scaffolding",
    "type",
    // Components
    "action-sheet",
    "backdrop",
    "bar",
    "tabs",
    "menu",
    "modal",
    "popover",
    "popup",
    "loading",
    "items",
    "list",
    "badge",
    "slide-box",
    "refresher",
    "spinner",
    // Forms
    "form",
    "checkbox",
    "toggle",
    "radio",
    "range",
    "select",
    "progress",
    // Buttons
    "button",
    "button-bar",

    // Util
    "grid",
    "util",
    "platform",
    // Animations
    "animations",
    "transitions";
```

如果你想要改动Ionicons，那么就去改*ionicons/ionicons.scss*；如果你想改动modal组件，那么就去改*_modal.scss*。 同理，当你想要改动动画的时候，就去改动*_animations.scss*。
在对Ionic进行改动之前，有两个文件你一定要理解清楚：

- _variables.scss
- _mixin.scss 就像他们的名字说的一样，他们分别存放了可以被重写的变量和可以被重用的混合式。如果你打开*_variables.scss*的时候， 你会在里面看到所有的颜色，字体，填充，边距，边界等等的变量。
  例如，你可以在里面搜索*button*文本，你将会找到一整块关于按钮如何设置的配置。随便摘抄其中一部分来看看：

  ```
  $button-positive-bg: $positive !default;
  $button-positive-text: #fff !default;
  $button-positive-border: darken($positive, 10%) !default;
  $button-positive-active-bg: darken($positive, 10%) !default;
  $button-positive-active-border: darken($positive, 10%) !default;
  ```

  在这里可以看到positive类是如何设置按钮的主题的。只要改动了*$position*变量一下，你就可以看到按钮是外观和感觉是如何随之更新的。
  另外一个范例是关于*Grid(*格子*)*的部分：

  ```
  // Grids
  // -------------------------------
  $grid-padding-width: 10px !default;
  $grid-responsive-sm-break: 567px !default; // smaller than landscape phone
  $grid-responsive-md-break: 767px !default; // smaller than portrait tablet
  $grid-responsive-lg-break: 1023px !default; // smaller than landscape tablet
  ```

  如果你想对格子的行为方式进行更改，那么就是改这里了。 同时，你可以参考这里的变量名对小，中，大尺寸的设备的设置媒体查询断点。
  你可以花费一些时间来了解这个文件以确认你可以重写哪些变量。截至目前为止，还没有任何关于SCSS变量和他的作用的官方文档。 但是*_variables.scss*文件里面的一些注释倒是很有帮助。
  接下来，打开*_mixin.scss*文件。这个文件是由Ionic组件使用的混合式组成的。例如，下面是*button-style*的混合式：

```
@mixin button-style($bg-color, $border-color, $active-bg-color,$active-border-colo
r, $color) {
  border-color: $border-color;
  background-color: $bg-color;
  color: $color;
  // Give desktop users something to play with
  &:hover {
      color: $color;
      text-decoration: none;
  }
  &.active,
  &.activated {
      border-color: $active-border-color;
      background-color: $active-bg-color;
      box-shadow: inset 0 1px 4px rgba(0,0,0,0.1);
  }
}
```

这个混合式里面有一个背景色，边框色，以及一个激活状态颜色，然后生成了*border-color*，*background-color*，*color*，*.hover*，*.active*以及*.activated*规则。
另一个用的比较多的混合式是*clearfix*：

```
@mixin clearfix {
  *zoom: 1;
  &:before,
  &:after {
      display: table;
      content: "";
      line-height: 0;
  }
  &:after {
      clear: both;
  }
}
```

他所做的是使row清晰化，这个类在很多地方用来管理布局。
再次声明，官方没有任何文档可以用来理解本文的混合式。

> 用到的名词
>
> - button 按钮
> - animation 动画
> - transition 过渡

使用变量与混合式

现在，我们学习过了Ionic框架的两个核心部分，接下来我们就要学会如何使用他们。
打开*_button.scss*。如果你记得的话，早先我们使用按钮组件的时候，我们添加的所有按钮样式或者按钮类型都是由*button*开头的。例如：

```
<button class="button button-positive button-block">Click Me</button>
```

*button*类为按钮组件提供了默认的样式。其他的类提供了修改的样式或者类型。多么了不起的一个CSS解耦方法，对吧？
回到咱们的文件，你将看到*button*类是如何设置的。在*button*类定义里面，我们可以看到很多的变量和混合式，这些我们之前见过一部分。
*button*类的一部分摘抄如下：

```
&.button-positive {
    @include button-style($button-positive-bg, $button-positive-border,$button-positiv
e-active-bg, $button-positive-active-border, $buttonpositive-text);
    @include button-clear($button-positive-bg);
    @include button-outline($button-positive-bg);
}
```

在这里有三个混合式用来生成一个*button-positive*的样式。这个样式将会在*button*和*button-positive*应用到元素的时候才会生效。
另一个需要查阅的文件是*_util.scss*。所有的工具类都会生成在这里，例如：*hide*， *show*，*padding*等等。
如果你想要对动画和过渡做任何变更的话，你可以去查看：*_animations.scss* 和 *_transitions.scss*。
这些文件名对于查找组件的SCSS代码帮助非常大。

SCSS工作流

现在我们知道了SCSS在哪里设置，如何设置的，我们可以使用下面的工作流在Ionic项目中使用SCSS。
步骤如下：

1. 设置好SCSS。
2. 打开*scss/ionic.app.scss*文件。
3. 在导入Ionic SCSS框架之前，添加/更新我们需要重写的变量。
4. 在导入Ionic SCSS框架之前，添加需要的字体。
5. 在导入Ionic SCSS框架之后，添加/重写预定义的类或者创建一个新的类。 然后，一个典型的自定义的*ionic.app.scss*文件就诞生了：

```
// Override or add variables
$positive: teal;
$custom: #aaa;
// add custom button variables
$button-custom-bg: $custom !default;
$button-custom-text: #eee !default;
$button-custom-border: darken($custom, 10%) !default;
$button-custom-active-bg: darken($custom, 10%) !default;
$button-custom-active-border: darken($custom, 10%) !default;
// define the ionic fonts path
$ionicons-font-path: "../lib/ionic/fonts" !default;
// import Ionic SCSS Framework
@import "www/lib/ionic/scss/ionic";
// build a custom button class that is specific to our app
.button-custom {
 /*
 Usage : <button class="button button-custom">
 Custom Styled Button
 </button>
 */
 @include button-style($button-custom-bg, $button-custom-border,
 $button-custom-active-bg, $button-custom-active-border, $button-customtext);
 @include button-clear($button-custom-bg);
 @include button-outline($button-custom-bg);
}
```

如果你想完全的改变Ionic app的外观和感觉，你可以从重写默认的颜色样本开始：

```
$light: #fff !default;
$stable: #f8f8f8 !default;
$positive: #387ef5 !default;
$calm: #11c1f3 !default;
$balanced: #33cd5f !default;
$energized: #ffc900 !default;
$assertive: #ef473a !default;
$royal: #886aea !default;
$dark: #444 !default;
```

然后你就可以发现你改的是哪一个组件。导航到正确的SCSS文件，查看对这个类有影响的变量。然后返回*ionic.app.scss*对他们进行重写。

> 用到的名词
>
> - markup 标记
> - reflect 反射
> - brand 个人品牌，牌子
> - teal 鸭屎蓝

创建一个样本

为了更好地理解之前的流程，我们将创建一个自己的主题，重写变量和类。我们将搭建一个侧边菜单app，然后对他默认的外观和感觉进行更改。
运行如下命令以创建一个新的侧边菜单app：

```
ionic start -a "Example 15" -i app.example.fifteen example15 sidemenu
```

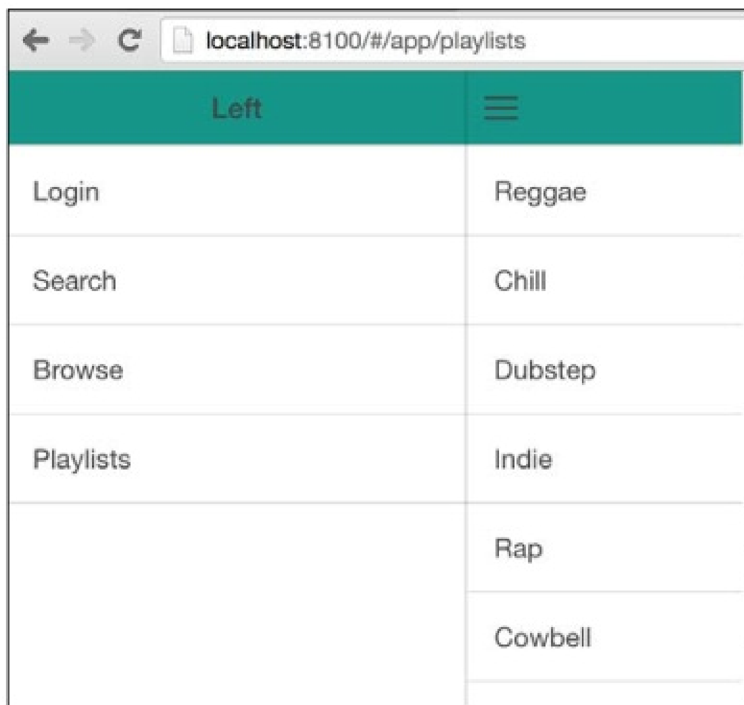
然后通过*cd*命令进入到*example15*文件夹，运行此命令：

```
ionic setup scss
```

这个命令将会为你的项目下载和设置SCSS依赖库。打开*ionic.app.scss*。此处的主旨是不更改任何标记或者添加任何新类，只是修改一些对于的变量来映射到新的主题。

> 这不是给你的应用定制主题的唯一途径。你也可以通过修改标记，添加新类以反射你的个人品牌。（例如 *button-mybrand*或者*bar-mybrand*），然后在SCSS里创建对应的变量和类。

我们将使用鸭屎蓝来给侧边菜单app指定主题。第一步我们要做的是修改*$stable*变量：
$stable: #009688
如果在添加上面的样式之后启动app的服务，你将看到页头已经变成了鸭屎蓝，如下：



页头的文本是黑色的。我们把他设为白色吧。打开*_bar.scss*，找到以下部分：

```
.title {
    color: #fff;
}
```

这个看起来不是个变量。所以，在这个案例中，我们将重写这个类。在*ionic.app.scss*中，在包含了Ionic CSS框架之后，我们添加以下代码：

```
.bar.bar-stable .title{
    color:#eee;
}
```

保存之后就可以即可看到标题文本颜色发生了改变。但是，你看到那个汉堡包菜单的文本还是黑色的。这个我们也想改掉。 当你打开浏览器开发者工具检查这个汉堡包菜单的时候，你会发现一个这样的按钮：

```
<button class="button button-icon button-clear ion-navicon" menutoggle="left"></button
>
```

在开发工具里面这个元素的样式里会看到*bar-stable button button-clear*这些类，他会将颜色设为#444。现在我们要追踪这个变量，然后重置他。
由于类的第一个部分是*.bar-stable*，所以这个可能是*_bar.scss*里面的定义。然后可以在*_bar.scss*里面找到：

```
.bar-stable {
.button {
    @include button-style($bar-stable-bg, $bar-stable-border,$bar-stable-active-bg, $b
ar-stable-active-border, $bar-stabletext);
    @include button-clear($bar-stable-text, $bar-title-font-size);
}
}
```

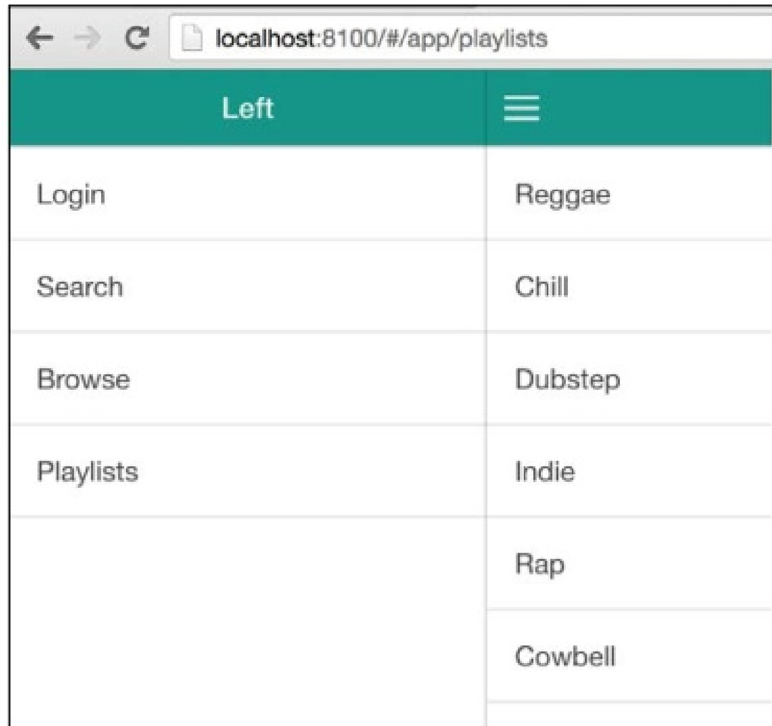
注意观察混合式*button-clear*。我们可以在*_mixins.scss*里面看到他的眼代码：

```
@mixin button-clear($color, $font-size:"") {
&.button-clear {
    border-color: transparent;
    background: none;
    box-shadow: none;
    color: $color;
    @if $font-size != "" {
        font-size: $font-size;
    }
}
&.button-icon {
    border-color: transparent;
    background: none;
}
}
```

我们可以看到，这个混合式的第一个参数名为*$color*。这样我们就知道我们需要重写哪个变量了。

回到*ionic.app.scss*文件，我们将重写*$bar-stable-text*变量：

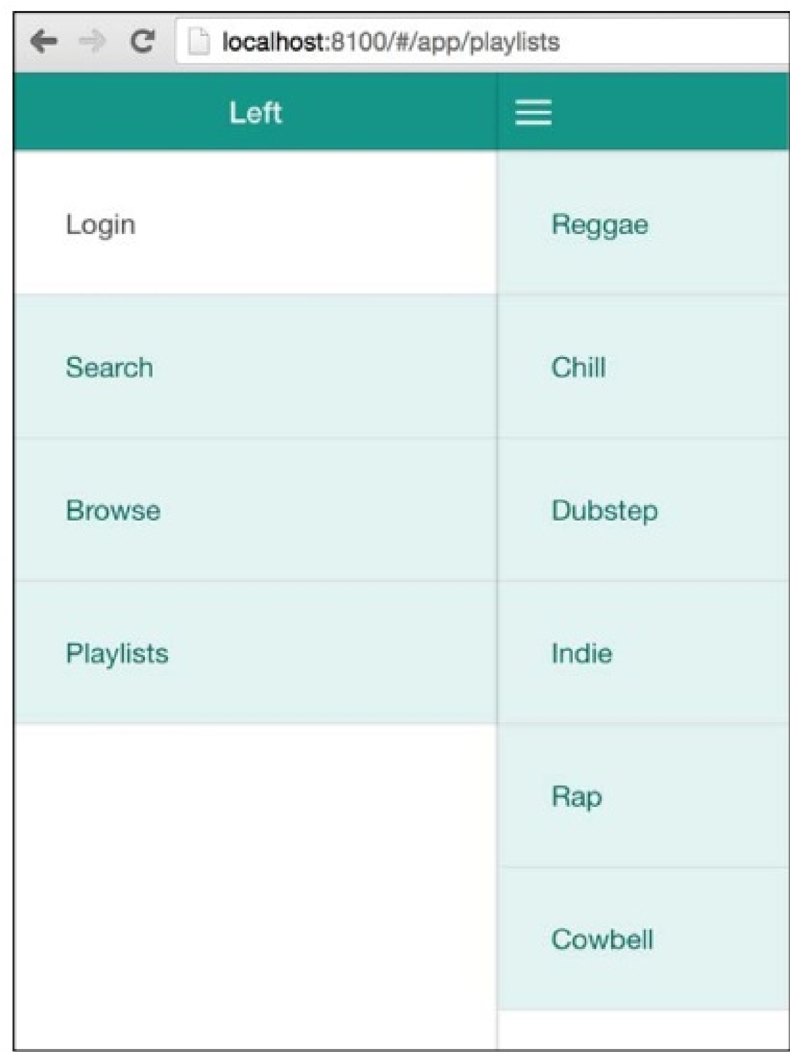
$bar-stable-text: #eee;
保存文件，然后就可以看到以下效果：



接下来，我们将更改字体的颜色。我们需要重写变量*$base-color*。现在我们要给列表条目增加更多的填充空间。我们再次打开*_items.scss*，找到这个： **padding: $item-padding;**
我们可以将*$item-padding*重写为30px。
接下来，为每个条目制作一个淡绿色背景。鉴于没有相关变量可用，我们需要去重写这个类：

```
.item-complex .item-content{
    background:#E0F2F1;
    color:#00695C;
}
```

保存文件之后，可以在浏览器里面看到如下效果：

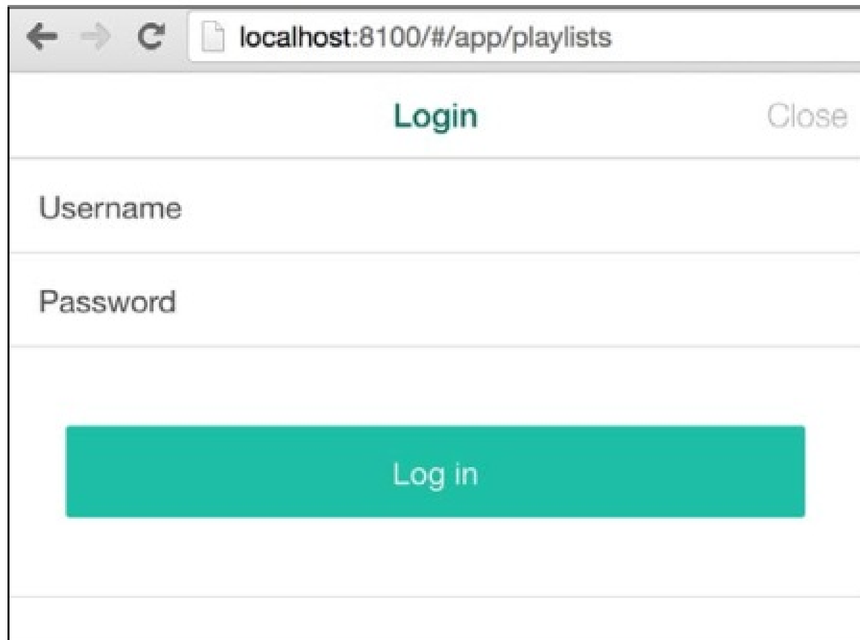


当点击登录连接的时候，将会出现一个modal弹出框。弹出框里面的**Login**按钮使用了positive类作为样式。使用以下值重写这部分：

```
$button-positive-bg: #00BFA5;
$button-positive-border: #00BFA5;
$button-positive-active-bg: #80CBC4;
$button-positive-active-border: #80CBC4;
$button-positive-text: #eee;
```

可以在*button-style*里找到所有*button-positive*使用的变量列表。这里我们处理了*button*类的两个状态。

Loginmodal看起来将会是这样的：



一切看起来都是那么的美好，除了列表条目的边框颜色和列表条目被选中之后的激活的背景色。我们现在就来处理这些。 打开*_items.scss*，找到*item-style*混合式，这个混合式接收的第二个参数是*$item-default-border*。这个参考将被用作边框色。
我们重写他为：

```
$item-default-border: #009688;
```

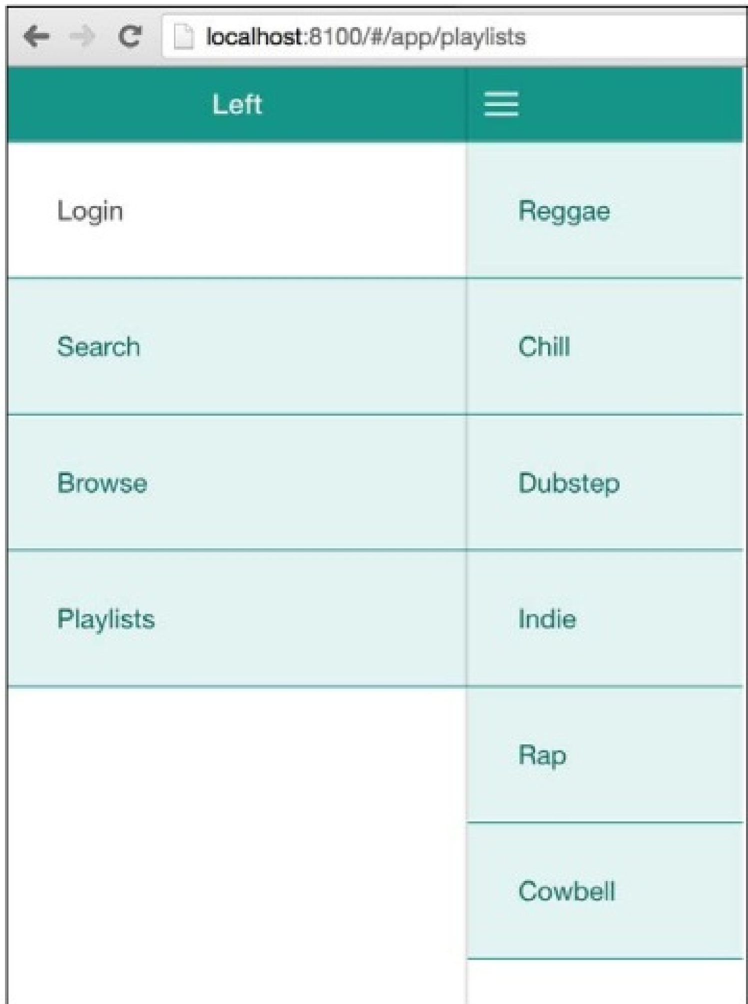
最后是激活背景色。在链接和按钮激活状态的部分，我们会看到一个*item-mixin-style*混合式。第一个参数名为*$item-default-active-bg*，这就是激活背景色了。 我们将在*ionic.app.scss*中重写他：

```
$item-default-active-bg: #B2DFDB;
```

完整的*ionic.app.scss*代码如下：

```
// override all $stable themed components
$stable: #009688;
// override the burger menu color
$bar-stable-text: #eee;
// override app base color
$base-color: #00695C;
// increase the item padding
$item-padding : 30px;
// override buttons
$button-positive-bg: #00BFA5;
$button-positive-border: #00BFA5;
$button-positive-active-bg: #80CBC4;
$button-positive-active-border: #80CBC4;
$button-positive-text: #eee;
// border & active bg color
$item-default-border: #009688;
$item-default-active-bg: #B2DFDB;
// The path for our ionicons font files, relative to the built CSS in www/css
$ionicons-font-path: "../lib/ionic/fonts" !default;
// Include all of Ionic
@import "www/lib/ionic/scss/ionic";
// Override title color
.bar.bar-stable .title{
    color:#eee;
}
// Override the item background and color
.item-complex .item-content{
    background:#E0F2F1;
    color:#00695C;
}
```

最终效果图如下：



这是一个基本的使用SCSS给你的Ionic app制定主题的范例。你可以给他添加更多的组件，然后给他们分别制定主题作为练习。

> 我给你展示了如何在Ionic SCSS设置中找到响应的变量和混合式。这个知识点可以用在任何重写变量和重用混合式来定制主题的Ionic SCSS项目。

总结

本章中，我们学习了如何定制Ionic app主题。我们由设置SCSS开始，然后浏览了Ionic SCSS结构。之后，我们通过重写变量以改变应用的主题。 最后，我们建立了一个app并且改变了他的外观。
接下来的章节里，我们将要浏览大量的Ionic指令和服务，他们将会帮助我们轻松的创建混合app。

第五章 Ionic指令与服务

回顾一下咱们到现在为止的旅程，我们从了解AngularJS作为一个客户端MVW框架的重要性。我们浏览了一下Apache Cordova，以及他是如何应用到整个混合应用开发栈里面来。 稍后，我们介绍了Ionic，解释了他是什么以及他是如何让我们轻松的开发混合应用的。 在第三章 *Ionic CSS*组件和导航中，我们看到了Ionic是如何在你的移动页面开发中用作一个唯一的CSS框架的， 在第四章 *Ionc*与*SCSS*中，我们学习了如何通过SCSS的能力去给组件定制主题。

在本章中，我们将要学习Ionic指令与服务，他们将给我们提供可重用的组件以及功能以帮助我们更快的开发应用。

看完本章，你将学会：

- Ionic指令
- Ionic服务

Ionic指令与服务

Ionic有纯CSS驱动的组件和需要借助JavaScript的魔力达到圆满的组件。由于Ionic使用AngularJS作为他的JavaScript框架， 所有的可重用的用户界面组件都会被写成指令形式，所有可重用的JavaScript功能片都将会被写成服务形式。
以下是一些Ionic指令的示例：

- 导航 *ion-nav-view*
- 页头与页脚 *ion-header-bar*和*ion-footer-bar*
- 列表 *ion-list*和*ion-item*
- 标签页 *ion-tabs*和*ion-tab*
- 侧边菜单 *ion-side-menus*和*ion-side-menu*

以下是一些Ionic服务的示例：

- 平台 *$ionicPlatform*
- 滚动 *$ionicScrollDelegate*
- 模态框 *$ionicModal*
- 导航条 *$ionicNavBarDelegate*
- 历史 *$ionicHistory*
- 弹出框 *$ionicPopup*

在接下来的部分中，我们将浏览一些Ionic指令和服务，理解如何使用他们。我们将根据需求交替选择使用Ionic核心指令和服务。

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通: https://github.com/learning-ionic/Chapter-5

> 用到的名词：
>
> - hook 钩子
> - action 动作

Ionic平台服务

第一个我们将要使用的服务是Ionic Platform服务(*$ionicPlatform*)。这个服务提供设备级别的的钩子，以使你更好的控制你的应用的行为。
我们从最基本的*ready*方法开始。这个方法会在设备准备好的时候，或者设备已经准备好的时候立即触发。
我们将新建一个项目来学习Ionic Platform服务。新建一个文件夹名为*chapter5*。在文件夹内打开命令行/终端，运行：

```
ionic start -a "Example 16" -i app.example.sixteen example16 blank
```

app创建好之后，打开*www/hs/app.js*找到以下部分：

```
.run(function($ionicPlatform) {
    $ionicPlatform.ready(function() {
    // Hide the accessory bar by default (remove this to show the  accessory bar above
 the keyboard
    // for form inputs)
    if(window.cordova && window.cordova.plugins.Keyboard) {
        cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
    }
    if(window.StatusBar) {
        StatusBar.styleDefault();
    }
    });
})
```

> 所有与Cordova有关的代码都需要写在*$ionicPlatform.ready*方法里面，因为这里是app生命链中插件初始化和预备使用的点。

可以看到*$ionicPlatform*服务作为依赖注入到了*run*方法。强烈建议在其他的AngularJS组件（如控制器，指令，这些都是你打算与Cordova交互的地方）中使用*$ionicPlatform.ready*方法。
在之前的*run*方法中，注意我们通过以下设置隐藏了键盘访问条：

```
cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
```

你可以重写这个设置为*false*。同时，注意表达式前面的*if*条件。在使用Cordova的变量之前进行检查永远是不会错的。

*$ionicPlatform*指令有一个侦查硬件返回按钮事件的简便的方法。一些（Android）设备有一个硬件返回按钮，如果你想监听返回按钮的按下时间， 那么你就要对*$ionicPlatform*服务的*onHardwareBackButton*方法进行关联了：

```
var hardwareBackButtonHandler = function() {
    console.log('Hardware back button pressed');
    // do more interesting things here
}
$ionicPlatform.onHardwareBackButton(hardwareBackButtonHandl
er);
```

这个时间需要在*$ionicPlatform.ready*方法中注册，最好是在AngularJS的*run*方法里面。*hardwareBackButtonHandler*回调函数将会在每次用户按下设备的返回按钮进行调用。

这个处理的的一个简单的逻辑功能是询问用户是否真的要退出app，以确保他们不会因为错误操作而退出app。

虽然有时候这很烦人。因此，你可以给用户提供一个设置，用来指定当他退出的时候是否需要显示这个提示。在此之上，你可以延迟注册此事件或者你可以退订此事件。

上述逻辑的完整代码如下：

```
.run(function($ionicPlatform) {
    $ionicPlatform.ready(function() {
        var alertOnBackPress = localStorage.getItem('alertOnBackPress');
        var hardwareBackButtonHandler = function() {
            console.log('Hardware back button pressed');
            // do more interesting things here
        }
        function manageBackPressEvent(alertOnBackPress) {
            if (alertOnBackPress) {
                $ionicPlatform.onHardwareBackButton(hardwareBackButtonHandler);
            } else {
                $ionicPlatform.offHardwareBackButton(hardwareBackButtonHandler);
            }
        }
        manageBackPressEvent(alertOnBackPress);
        // later in the code/controller when you let
        // the user update the setting
        function updateSettings(alertOnBackPressModified) {
            localStorage.setItem('alertOnBackPress', alertOnBackPressModified);
            manageBackPressEvent(alertOnBackPressModified)
        }
    });
})
// when the app boots up
```

在上面的代码片段里，我们在*localStorage*里面查找*alertOnBackPress*的值。接下来，我们创建了一个名为*hardwareBackButtonHandler*的函数，这个函数将在返回按钮按钮的时候触发。最后一个名为*manageBackPressEvent()*的工具方法接收一个布尔值，这个布尔值只是是否注册*HardwareBackButton*的回调或者移除注册。
有了这些设置，当app启动的时候，我们使用从*localStorage*读取的值调用了*manageBackPressEvent*方法。如果在*localStorage*中这个值存在并且为*true*的时候，我们就注册这个事件； 反之则不注册。稍后，我们可以给用户提供一个设置控制器以对这个设置进行更改。当用户改变了*alertOnBackPress*的状态时，我们调用了*updateSettings*方法并传入了用户是否需要被提示。 *updateSettings*更新了*localStorage*里面的设置并且调用了*manageBackPressEvent*方法。
这是一个很强大的范例，展示了当AngularJS与Cordova组合起来，提供API供你轻松的管理你的应用。

> 这个范例第一眼看起来也许会有点复杂，但是大部分你需要消化的服务都比较类似。你需要根据情况与优先级对事件进行注册和移除注册。所以我觉得在这里放出这个范例是再合适不过了， 这个想法在你完成本章学习之后会更加确定。

registerBackButtonAction

*$ionicPlatform*提供了另一个名为*registerBackButtonAction*的方法。当你在按下了返回按钮你想要控制你的应用的行为的另一个API。
默认按下返回按钮只会执行一个任务。例如，当你有一个多页面应用，然后你从page1导航到了page2，这个时候你点击了返回按钮，你就会返回page1. 在另一个场景下，当用户从page1导航到page2，而page2加载的时候弹出了一个对话框，此时按下返回按钮，只会隐藏弹出框，而不会导航到page1.
*registerBackButtonAction*方法提供了一个钩子去重写这个行为。
*registerBackButtonAction*接收以下3个参数：

- callback：这个是事件触发的时候将会调用的方法
- priority：这个是事件监听器的优先级
- actionId（可选）：这个是分配给这个动作的ID 默认的优先级有以下几种：
- 预览视图 = 100
- 关闭侧边菜单 = 150
- 清理modal = 200
- 关闭动作表单= 300
- 清理弹出框 = 400
- 清理加载遮盖层 = 500 因此，当你想要重写返回按钮的默认行为的时候，你需要这样子去做：

  ```
  var cancelRegisterBackButtonAction =
  $ionicPlatform.registerBackButtonAction(backButtonCustomHandler,201);
  ```

> 这个监听器将重写（抢夺优先权）所有优先级低于201的默认的监听器，也就是清理modal，关闭侧边菜单和预览视图，但是他不会重写优先级高于他的其他的监听器。*$ionicPlatform.registerBackButtonAction*执行的时候，返回一个函数。我们已经将这个函数指派给了*cancelRegisterBackButtonAction*变量。 执行*cancelRegisterBackButtonAction*移除*registerBackButtonAction*监听器的注册。

on方法

除了上面方法之外，*$ionicPlatform*有一个通用的*on*方法可以用来监听所有的Cordova事件：https://cordova.apache.org/docs/en/edge/cordova_events_events.md.html
你可以为应用的*pause*（暂停）, *application resume*（重回）,*volumedownbutton*（音量加）,*volumupbutton*（音量减）等等设置钩子，然后根据情况执行自定义的方法。
可以这样在*$ionicPlatform.ready*方法内设置这些监听器：

```
var cancelPause = $ionicPlatform.on('pause', function() {
    console.log('App is sent to background');
    // do stuff to save power
});
var cancelResume = $ionicPlatform.on('resume', function() {
    console.log('App is retrieved from background');
    // re-init the app
});
// Supported only in BlackBerry 10 & Android
var cancelVolumeUpButton = $ionicPlatform.on('volumeupbutton',function() {
    console.log('Volume up button pressed');
    // moving a slider up
});
var cancelVolumeDownButton = $ionicPlatform.on('volumedownbutton',function() {
    console.log('Volume down button pressed');
    // moving a slider down
});
```

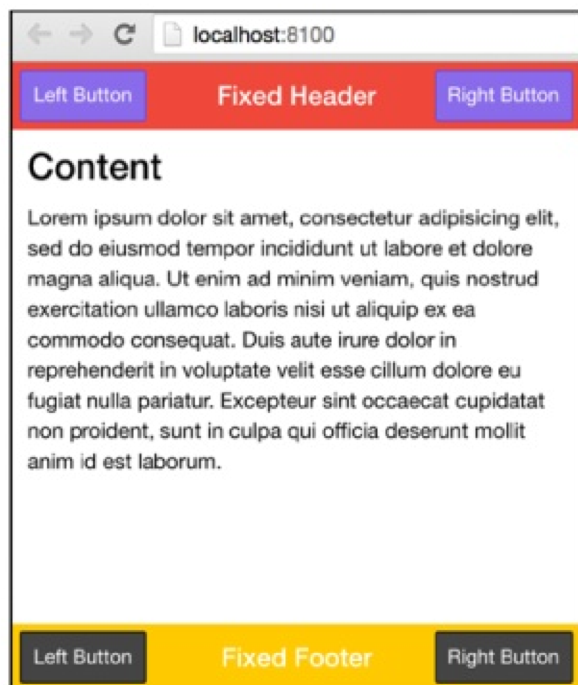
*on*方法返回一个当移除事件监听需要的函数。
现在你应该知道在处理移动OS事件和硬件按钮的时候如何更好的控制你的app。

页头与页脚

使用*ion-header-bar*和*ion-footer-bar*指令，可以给你的app添加一个固定的页头和页脚。 范例结构如下：

```
<ion-header-bar align-title="center" class="bar-assertive">
    <div class="buttons">
        <button class="button button-royal" ngclick="doSomething()">Left Button</butto
n>
    </div>
    <h1 class="title">Fixed Header</h1>
    <div class="buttons">
        <button class="button button-royal">Right Button</button>
    </div>
</ion-header-bar>
<ion-content>
    <div class="padding">
        <h3>Content</h3>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing
        elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna aliqua. Ut
        enim ad minim veniam,
        quis nostrud exercitation ullamco laboris nisi ut
        aliquip ex ea commodo
        consequat. Duis aute irure dolor in reprehenderit in
        voluptate velit esse
        cillum dolore eu fugiat nulla pariatur. Excepteur sint
        occaecat cupidatat non
        proident, sunt in culpa qui officia deserunt mollit
        anim id est laborum. </p>
    </div>
</ion-content>
<ion-footer-bar align-title="left" class="bar-energized">
    <div class="buttons">
        <button class="button button-dark">Left Button</button>
    </div>
    <h1 class="title">Fixed Footer</h1>
    <div class="buttons" ng-click="doSomething()">
        <button class="button button-dark">Right Button</button>
    </div>
</ion-footer-bar>
```

结果如下：



内容 Content

接下来将要学习的是内容相关的指令。从*ion-content*指令开始。

ion-content

*ion-content*用作一个内容显示区域。这条指令有很多选择可以供你更好的控制内容显示区域。你可以对*ion-content*使用Ionic自定义的滚动视图或者使用浏览器默认的覆盖层滚动。这本书写作的时候，*ion-content*指令包含的所有属性如下：

```
<ion-content
    delegate-handle=""
    direction=""
    locking=""
    padding=""
    scroll=""
    overflow-scroll=""
    scrollbar-x=""
    scrollbar-y=""
    start-x=""
    start-y=""
    on-scroll=""
    on-scroll-complete=""
    has-bouncing=""
    scroll-event-interval="">
    <h1>Heading!</h1>
</ion-content>
```

*ion-content*关键的几个属性解释如下：

- scroll：决定是否使用滚动，默认：true
- overflow-scroll：使用浏览器的覆盖层滚动
- on-scroll：当显示内容滚动的时候调用的函数/表达式
- on-scroll-complete：当显示内容滚动完成之后执行的函数/表达式
- scroll-event-interval：在调用on-scroll之前的定时器，默认：10ms
- scrollbar-x：展示水平滚动条，默认：true
- scrollbar-y：展示垂直滚动条，默认：true
- locking：锁住滚动，默认：true
- direction：只是如何滚动，可用：x，y（默认），xy
- has-bouncing：允许你滚动超出显示内容的边界，默认：iOS上true， Android上false

ion-scroll

也可以使用*ion-scroll*指令来控制显示内容的滚动。这个指令用于替换*ion-content*指令的。使用方式也是非常简单的：

```
<ion-view ng-controller="MyAppCtrl" cache-view="false">
    <ion-scroll zooming="true" direction="xy" style="width: 300px; height: 300px">
        <div style="width: 1000px; height: 1000px; backgroundcolor:teal"></div>
    </ion-scroll>
</ion-view>
```

> 注意一点：需要将scroll盒的高度设置为内部需要滚动的内容的高度。 如果想要使滚动区域居中，可以使用:ion-content

后面将会谈到*ion-view*的，不要着急。

ion-refresher

另一个能够方便管理显示内容的指令是*ion-refresher*。这个是添加到滚动视图（*ion-content*或者*ion-scroll*）中用作下拉更新的指令。
新建一个空白模板项目来测试：

```
ionic start -a "Example 17" -i app.example.seventeen example17 blank
```

使用*cd*命令，进入到*example17*文件夹，运行如下命令：

```
ionic serve
```

这个范例中，我们将实现一个下拉更新的功能。我们将创建一个工厂然后伪造HTTP请求并返回一个对象数组。数组对象是一个伪哈希，有两个用于显示的属性。
这个工厂将会从*AppCtrl*调用，因为*AppCtrl*是视图的控制器。这个控制器会实现一个*doRefresh*方法，这个方法在用户进行下拉更新操作的时候调用。 这个方法将直接跟工厂对话，拿到随机数据，然后这些数据将会被添加到当前的显示列表。
我们将在*www/js/app.js*的*config*方法后面定义如下数据工厂：

```
.factory('DataFactory', function($timeout, $q) {
    var API = {
        getData: function(count) {
            // Spoof a network call using promises
            var deferred = $q.defer();
            var data = [],
            _o = {};
            count = count || 3;
            for (var i = 0; i < count; i++) {
                _o = {
                    // http://stackoverflow.com/a/8084248/1015046
                    random: (Math.random() + 1).toString(36).substring(7),
                    time: (new Date()).toString().substring(15, 24)
                };
                data.push(_o);
            };
            $timeout(function() {
                // success response!
                deferred.resolve(data);
            }, 1000);
            return deferred.promise;
        }
    };
    return API;
})
```

> 以上范例可简单使用settiemout替代promise就可以了。但是，由于AngularJS严重依赖promise驱动，所以我们这个范例使用的是promise，我个人也认为应当这么做。

在上面的工厂中，我们用到了AngularJS的*$q*服务，这个服务帮助我们以异步的方式返回结果，类似HTTP请求。 我们创建的返回结果对象包含一个随机字母的字符串作为第一个属性，以日期子串作为第二属性。这个仅做显示用途，没别的意义。
在*www/js/app.js*中，我们的控制器代码如下：

```
.controller('AppCtrl', function($scope, DataFactory) {
    $scope.items = [];
    $scope.doRefresh = function() {
        DataFactory.getData(3)
        .then(function(data) {
            // extend the $scope.items array with the response
            // array from getData();
            // http://stackoverflow.com/a/1374131/1015046
            Array.prototype.push.apply($scope.items, data);
        }).finally(function() {
            // Stop the ion-refresher from spinning
            $scope.$broadcast('scroll.refreshComplete');
        });
    }
    // load data on page load
    DataFactory.getData(3).then(function(data) {
        $scope.items = data;
    });
})
```

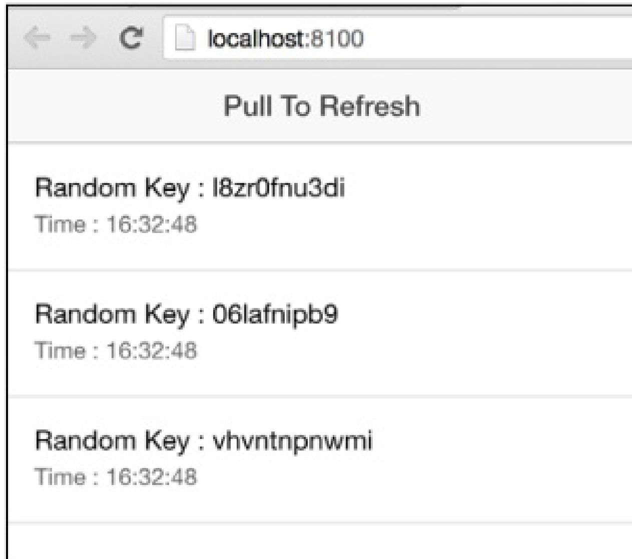
在上面的控制器代码中，我们给控制器注入了DataFactory作为依赖。我们可以在控制器初始化之后调用*getData*方法。*getData*方法将会返回3条记录，这3条记录我们将指派给列表显示。
接下来，我们定义了用户下拉更新的时候需要的*doRefresh*方法。这个方法也使用了*DataFactory*的*getData*方法并返回了他的返回值。 然后我们将*getData*方法的返回值数组附加到*scope*变量上。
注意，我们是把数据附加到已有的数组的后面的。如果在用户需要在顶部查看最新数据的时候，那么数据将要前置到已有数组。 最后，我们广播了*scroll.refreshComplete*事件，这样将会隐藏刷新按钮/微调（spinner）。
现在可以更新*www/index.html*的body部分了：

```
<body ng-app="starter" ng-controller="AppCtrl">
    <ion-header-bar class="bar-stable">
        <h1 class="title">Pull To Refresh</h1>
    </ion-header-bar>
    <ion-content>
        <ion-refresher pulling-text="Pull to refresh..." onrefresh="doRefresh()">
        </ion-refresher>
        <ion-list>
            <ion-item collection-repeat="item in items">
                <h2>Random Key : {{item.random}}</h2>
                <p>Time : {{item.time}}</p>
            </ion-item>
        </ion-list>
    </ion-content>
</body>
```

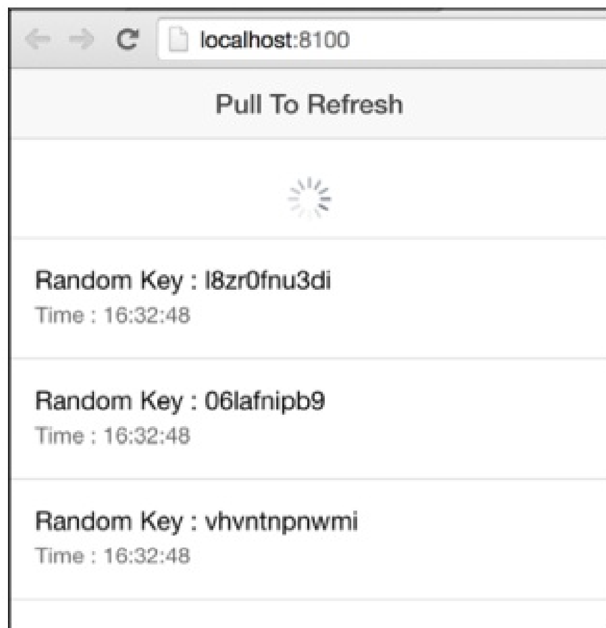
注意看，我们将*ion-refresher*作为*ion-content*的直接子元素添加进来了。然后我们使用*collection-repeat*替换了*ng-repeat*。

> *collection-repeat*比*ng-repeat*在显示超大列表的时候效率更高。更多信息关于*collection-repeat*请参考： http://ionicframework.com/docs/api/directive/collectionRepeat/

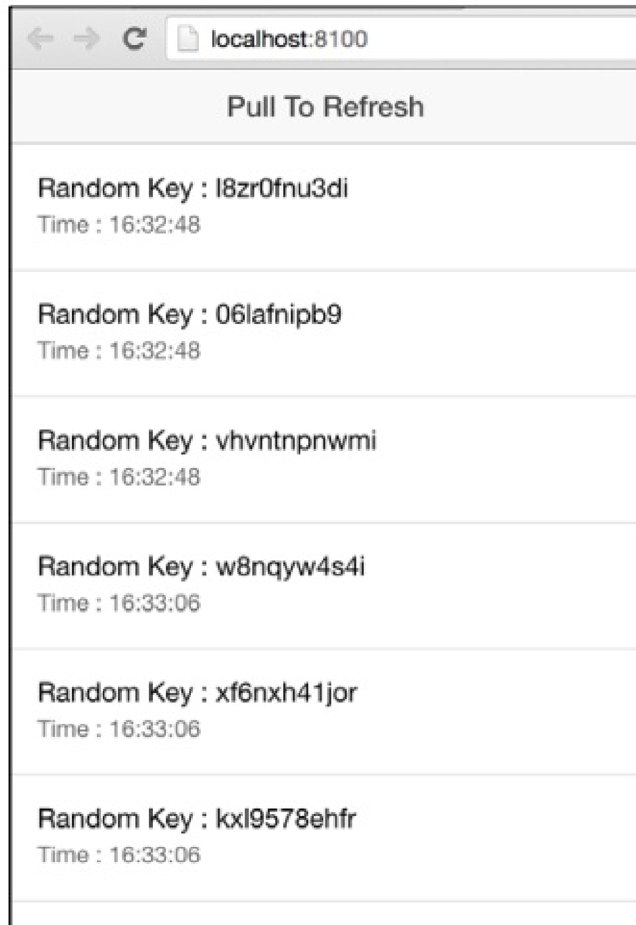
保存所有文件，然后你可以在浏览器中看到如下画面：



使用鼠标下拉页面的时候，将会看到这样子的画面：



一旦promise完成之后，数据将会立即返回控制器然后附加到条目数组里。所以这些流程完成之后，将会触发 *scroll.refreshComplete*事件，隐藏加载图标/spinner。更新后的页面如下：



> 你也可以通过*ion-refresher*的pulling-text, pulling-iocn, spinner属性设置下拉刷新的文本，图标和spinner。其他可用选择，请参考：
> http://ionicframework.com/docs/api/directive/ionRefresher/

ion-infinite-scroll

Ionic提供了另一个方便的名为*ion-infinite-scroll*的指令，和*ion-refresher*类似。当用户达到列表的末尾的时候，这个指令将会调用和上面例子里面*doRefresh*方法类似的方法来更新列表。*ion-refresher*与*ion-infinite-scroll*的不同点是当用户明确的需要加载一个列表的时候使用*ion-refresher*，而*ion-infinite-scroll*则是当用户滚动的时候自动更新列表。
想要在之前的范例中使用*ion-infinite-scroll*，只需要将body部分更新为如下即可：

```
<body ng-app="starter" ng-controller="AppCtrl">
    <ion-header-bar class="bar-stable">
        <h1 class="title">Pull To Refresh</h1>
    </ion-header-bar>
    <ion-content>
        <ion-refresher pulling-text="Pull to refresh..." onrefresh="doRefresh()">
        </ion-refresher>
        <ion-list>
            <ion-item collection-repeat="item in items">
                <h2>Random Key : {{item.random}}</h2>
                <p>Time : {{item.time}}</p>
            </ion-item>
         </ion-list>
        <ion-infinite-scroll on-infinite="loadMore()"  distance="1%">
        </ion-infinite-scroll>
    </ion-content>
</body>
```

注意*ion-infinite-scroll*是加载列表后面的，他有一个属性名为*on-infinite*。当距离是1%的时候，将会评估这个属性。 同时我们也把*ion-refresher*留下来了。因为这是一个例子而已，我想要给你展示如何在一个例子中同时使用*ion-refresher*和 *ion-infinite-scroll*。

在一个实时新闻馈送app中，你也许会这么做。当用户下拉更新的时候，你会要给用户展示最新的新闻，当用户滚动到下面的时候，你需要给用户展示旧新闻。

更新了*loadMore*的新版本的*AppCtrl*如下：

```
.controller('AppCtrl', function($scope, DataFactory) {
    $scope.items = [];
    $scope.doRefresh = function() {
        DataFactory.getData(3)
        .then(function(data) {
            // extend the $scope.items array with the response
            // array from getData();
            // http://stackoverflow.com/a/1374131/1015046
            Array.prototype.push.apply($scope.items, data);
        }).finally(function() {
            // Stop the ion-refresher from spinning
            $scope.$broadcast('scroll.refreshComplete');
        });
    }
    $scope.loadMore = function() {
        DataFactory.getData(3)
        .then(function(data) {
            // extend the $scope.items array with the response
            // array from getData();
            // http://stackoverflow.com/a/1374131/1015046
            Array.prototype.push.apply($scope.items, data);
        }).finally(function() {
            // Stop the ion-refresher from spinning
            $scope.$broadcast('scroll.infiniteScrollComplete');
        });
    }
    // load data on page load
    DataFactory.getData(3).then(function(data) {
     $scope.items = data;
    });
})
```

我们给scope变量添加了*loadMore*方法。这个基本和*doRefresh*方法差不多，除了我们在这里广播的事件是*scroll.infiniteScrollComplete*。
保存文件，回到浏览器，你将看到当一部分数据隐藏的时候*ion-infinite-scroll*会开始调用方法了。
当你下拉刷新的时候，列表会再次更新。但是请记住，我们并不是前置列表，所以数据还是添加到末尾的。

ionicScrollDelegate

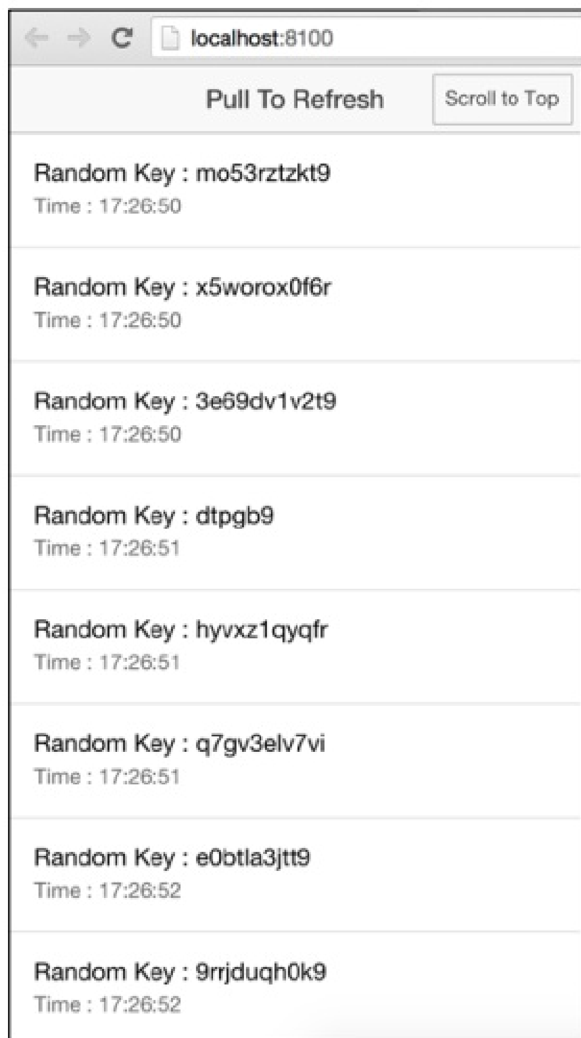
Ionic提供了一个控制滚动视图的服务。名为*$ionicScrollDelegate*。*$ionicScrollDelegate*服务提供了一系列的API，用于滚动，缩放以及取得当前滚动位置等等。
在之前的范例中，我们可以在页头里面添加一个按钮名为*Scroll To Top*。在用户下拉过列表之后，当用户点击这个，我们将通过*$ionicScrollDelegate*将他滚动到顶部。
更新后的*ion-header-bar*是这样的：

```
<ion-header-bar class="bar-stable">
    <h1 class="title">Pull To Refresh</h1>
    <button class="button" ng-click="scrollToTop()">Scroll to Top</button>
</ion-header-bar>
```

然后在控制器中，我们在注入了*$ionicScrollDelegate*之后，添加了*scrollToTop*方法定义：

```
$scope.scrollToTop = function() {
    $ionicScrollDelegate.scrollTop();
}
```

现在，无论用户把列表拉到多下面，用户都可以通过按下这个按钮返回列表顶部，如下：



> 更多信息关于*$ionicScrollDelegate*请参考：
> http://ionicframework.com/docs/api/service/$ionicScrollDelegate

导航

接下来出场的是导航相关的组件。导航组件有许许多都的指令和许许多多的服务。
第一个出场的指令是*ion-nav-view*。
在第三章 *Ionic CSS*组件和导航中，我们已经学习过Ionic状态路由器并了解了他是如何工作的。我们也看过了Ionic中的*ion-nav-view*是如何向AngularJS里面的*ui-view*一样工作的。
当app启动的时候，*$stateProvider*将会查找默认的状态，然后尝试加载*ion-nav-view*里面对应的模板。

ion-view

接下来出场的是*ion-view*指令。*ion-view*是*ion-nav-view*的子类。这个指令用来作为添加页头信息和其他内容的容器。当应用状态改变的时候，会在父容器*ion-nav-view*内展示对应的视图。
在Ionic中，为改善执行效率，视图都会被缓存起来。当视图离开*ion-nav-view*之后，他的子元素都将从DOM中移除，他的scope也将和*$watch*循环断开连接。 当之前缓存的视图进入*ion-nav-view*的时候，他的scope将会重新连接，已有的元素都将重新激活。
新建一个空白模板项目来学习：

```
ionic start -a "Example 18" -i app.example.eighteen example18 blank
```

通过*cd*目录，进入到*example18*文件夹，然后运行：

```
ionic serve
```

接下来，我们将给这个app添加一个路由器，这个路由器有2个状态。打开*www/js/app.js*在*run*方法后添加以下方法：

```
.config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
    .state('page1', {
        url: '/page1',
        templateUrl: 'page1.html'
    })
    .state('page2', {
        url: '/page2',
        templateUrl: 'page2.html'
    });
    $urlRouterProvider.otherwise('/page1');
})
```

我们分别给*page1*和*page2*创建了两个状态。接下来，修改*www/index.html*的body部分：

```
<body ng-app="starter">
    <ion-nav-bar></ion-nav-bar>
    <ion-nav-view></ion-nav-view>
    <!-- Templates -->
    <script type="text/ng-template" id="page1.html">
        <ion-view view-title="Title">
            <ion-content>
            <h3>Page 1</h3>
            <button class="button button-dark" ui-sref="page2">Navigate to Page 2</but
ton>
            </ion-content>
        </ion-view>
    </script>
    <script type="text/ng-template" id="page2.html">
        <ion-view view-title="Title">
            <ion-content>
            <h3>Page 2</h3>
            <button class="button button-dark" ui-sref="page1"> Navigate to Page 1</bu
tton>
            </ion-content>
        </ion-view>
    </script>
</body>
```

我们创建了一个供*ion-view*内容注入的*ion-nav-view*指令。我们的视图是通过*script*标签语法创建的。这样一来，AngularJS就不用发起AJAX请求来加载模板来。 但是这么做对于开发者和维护者来说都是很不友好的。

*ion-view*指令有一个子指令*ion-content*。每次导入新模板的时候，他都会进行缓存。你可以通过更改*ion-view*的属性来控制视图状态的外观，同理也可以控制缓存行为。

例如，在以上代码中，我们给*ion-view*标签添加一个*view-title*属性。这个值是用作页面标签，同时在没有*ion-nav-bar*的情况下，也可以用作导航条标题。

如果想要禁用模板缓存，在*ion-view*上面设置*cache-view*属性为*false*就可以了。同时也可以设置*hide-nav-bar*来控制是否显示*nav-bar*。

加上上面这些属性之后，*ion-view*看起来差不多是这样子的：

```
<ion-view view-title="Title" cache-view="false" hide-navbar="false" hide-back-button="
true" can-swipe-back="false">
```

为避免在导航条里面显示返回按钮，我们也可以控制返回按钮的显示。在后续学习*ion-nav-bar*的时候会学到这一点。

运行以下命令，可以验证目前所有对*example18*的更改：

```
ionic serve
```

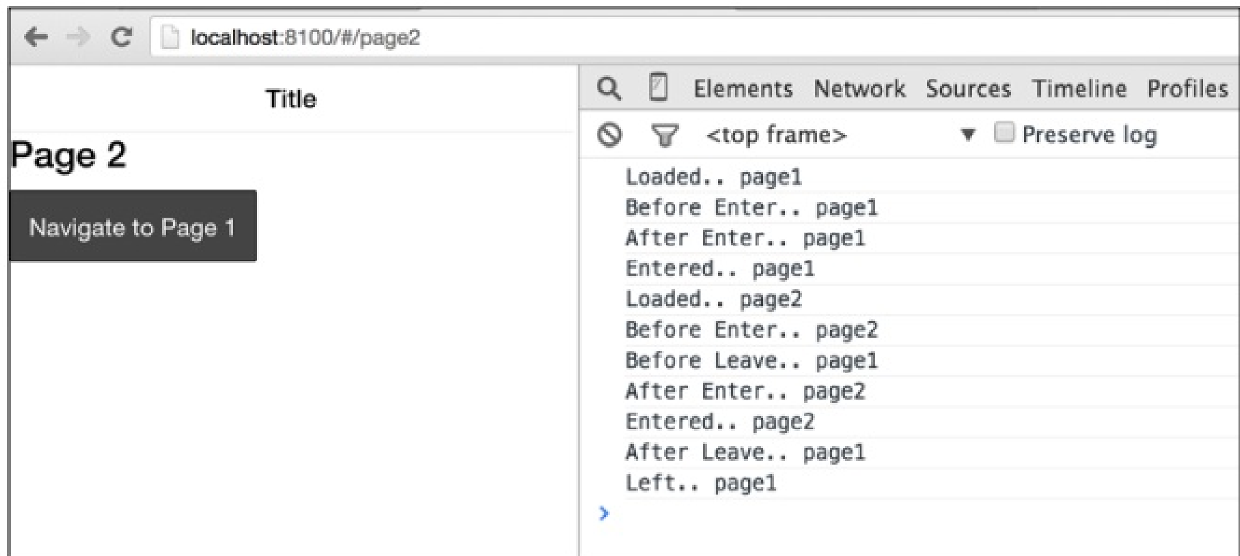
Ionic视图事件

Ionic视图有很多生命周期方法，你可以在这些方法里添加你的自定义行为。我们将给*example18*添加一个*run*方法，代码如下。 在*run*方法中，我们给*$ionicview*添加了事件监听器。打开*www/js/app.js*添加以下代码：

```
.run(function($ionicPlatform, $rootScope) {
    // View Life Cycle
    $rootScope.$on('$ionicView.loaded', function(event, view) {
        console.log('Loaded..', view.stateName);
    });
    $rootScope.$on('$ionicView.beforeEnter', function(event, view)
    {
        console.log('Before Enter..', view.stateName);
    });
    $rootScope.$on('$ionicView.afterEnter', function(event, view)
    {
        console.log('After Enter..', view.stateName);
    });
    $rootScope.$on('$ionicView.enter', function(event, view) {
        console.log('Entered..', view.stateName);
    });
    $rootScope.$on('$ionicView.leave', function(event, view) {
        console.log('Left..', view.stateName);
    });
    $rootScope.$on('$ionicView.beforeLeave', function(event, view)
    {
        console.log('Before Leave..', view.stateName);
    });
    $rootScope.$on('$ionicView.afterLeave', function(event, view)
    {
        console.log('After Leave..', view.stateName);
    });
    $rootScope.$on('$ionicView.unloaded', function(event, view) {
        console.log('View unloaded..', view.stateName);
    });
})
```

> 你可以给一个单独的模块添加多个*run*方法。

这样，在我们从page1导航到page2的时候，你将会看到下面这样的结果：



你可以追踪这些事件的触发顺序，然后根据需求给他们绑定你的自定义行为。

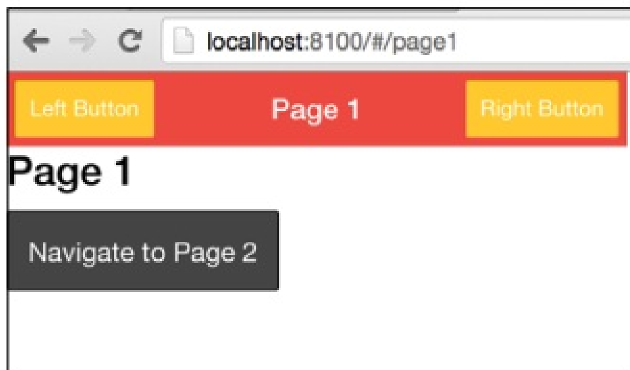
> *$ionicView.loaded*和*$ionicView.unloaded*只会在控制器的*$rootScope*里生效，*$scope*里面不会有任何效果。其他的方法可以。

ion-nav-bar

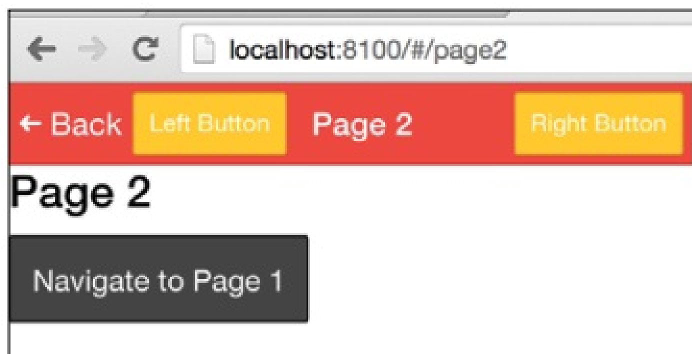
当我们制作多页面应用的时候,*ion-nav-bar*将会是你的好基友。这个指令负责在你的状态改变的时候更新页面标题栏。在*example18*中，我们添加了一个简化版的*ion-nav-bar*。
现在，我们来看一下稍微强化版的。在*www/index.html*中，使用以下代码替换*ion-nav-bar*：

```
<ion-nav-bar class="bar-assertive">
    <ion-nav-buttons side="left">
        <button class="button button-energized" ng-click="leftyClick()">
        Left Button
        </button>
    </ion-nav-buttons>
    <ion-nav-back-button class="button-clear">
        <i class="ion-arrow-left-c"></i> Back
    </ion-nav-back-button>
    <ion-nav-buttons side="right">
        <button class="button button-energized" ng-click="rightyClick()">
        Right Button
        </button>
    </ion-nav-buttons>
</ion-nav-bar>
```

在这段代码中，你可以看到*ion-nav-bar*使用*ion-nav-buttons*或者*ion-nav-back-button*作为子指令。
*ion-nav-buttons*用于将按钮展示在页头导航条。更新后的页面效果如下：



你可以定义*leftyClick()*和*rightyClick()*功能以供按钮点击的时候调用。也可以添加一个Header Controller并定义这些方法，然后将Header Controller添加给*ion-nav-bar*, 或者也可以在root scope里面定义*leftyClick()*和*rightyClick()*，虽然这个做法不怎么理想化。在实际环境中，根据情况右上角一般显示的是**Logout**按钮，**Option**按钮或者**Add**按钮。
*ion-nav-bar*也是*ion-nav-back-button*的宿主。这个指令负责在页面之间导航的时候自动显示返回按钮：



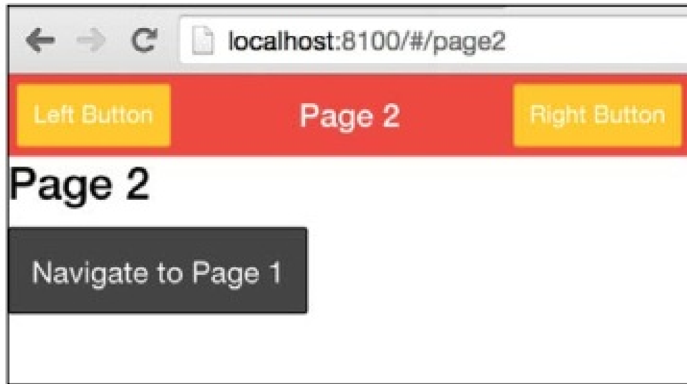
看到了吗，返回按钮自动出现了，把左边的按钮从原先的位置挤走了。

> *ion-nav-bar*指令只有在模板内容被包装在*ion-view*标签里的时候才会正常工作。

那么，这样我们就又回到了*ion-view*标签的属性来。你可以设置*ion-view*的*hide-nav-bar*属性为*false*；这样将会为当前页面显示导航。或者你可以设置*back-button*为*false*以隐藏页面上的返回按钮。
修改后的page2的*ion-view*如下：

```
<ion-view view-title="Page 2" hide-nav-bar="false" hide-backbutton="true">
```

此时，当你从page1导航只page2的时候，你会发现返回按钮不见了：



ion-nav-buttons

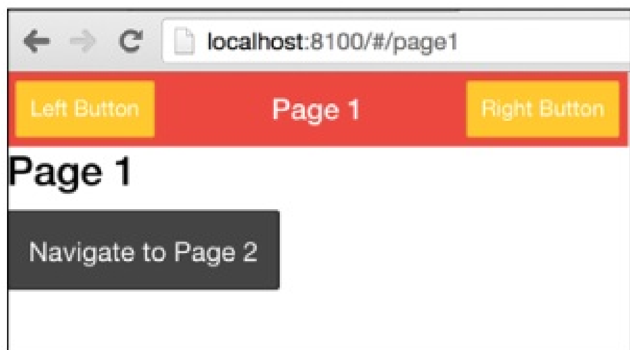
Ionic对页头里面的按钮提供了细粒度的控制。如果你在*ion-view*中声明了*ion-nav-buttons*，在模板中，他们将会覆盖*ion-nav-bar*指令里面的。
我们将*page2*模板更新如下：

```
<script type="text/ng-template" id="page2.html">
    <ion-view view-title="Page 2" hide-nav-bar="false" hide-back-button="true">
        <ion-nav-buttons side="left">
            <button class="button button-calm" ng-click="settingsClick()">
            Settings
            </button>
        </ion-nav-buttons>
        <ion-nav-buttons side="right">
            <button class="button button-calm" ng-click="optionsClick()">
            Options
            </button>
        </ion-nav-buttons>
        <ion-content>
            <h3>Page 2</h3>
            <button class="button button-dark" ui-sref="page1">
            Navigate to Page 1
            </button>
        </ion-content>
    </ion-view>
</script>
```

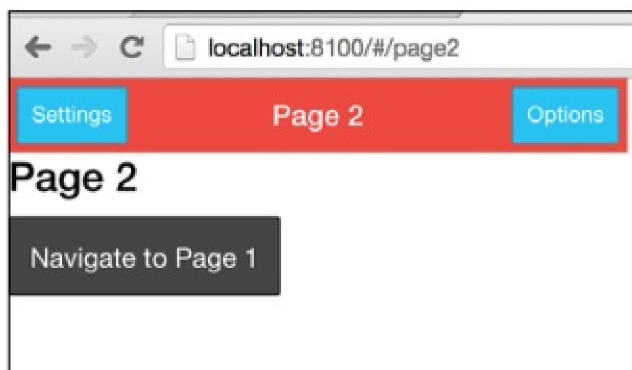
记住，我们不是对*ion-nav-bar*里面的*ion-nav-buttons*进行更改：

```
<ion-nav-bar class="bar-assertive">
    <ion-nav-buttons side="left">
        <button class="button button-energized" ng-click="leftyClick()">
        Left Button
        </button>
    </ion-nav-buttons>
    <ion-nav-back-button class="button-clear">
        <i class="ion-arrow-left-c"></i> Back
    </ion-nav-back-button>
    <ion-nav-buttons side="right">
        <button class="button button-energized" ng-click="rightyClick()">
        Right Button
        </button>
    </ion-nav-buttons>
</ion-nav-bar>
```

保存文件回到浏览器，page1效果如下：



page2在模板内显示了*ion-nav-button*：



$ionicNavBarDelegate

可以在控制器内控制*ion-nav-bar*，*$ionicNavBarDelegate*服务也可以。
为了更好的理解，我们给模板添加两个控制器：page1.html的*PageOneCtrl*和page2.html的*PageTwoCtrl*。
更新后的模板如下：

```
<script type="text/ng-template" id="page1.html">
    <ion-view ng-controller="PageOneCtrl">
        <ion-content>
            <h3>Page 1</h3>
            <button class="button button-dark" ui-sref="page2">
            Navigate to Page 2
            </button>
        </ion-content>
    </ion-view>
</script>
<script type="text/ng-template" id="page2.html">
    <ion-view ng-controller="PageTwoCtrl">
        <ion-content>
            <h3>Page 2</h3>
            <button class="button button-dark" ui-sref="page1">
            Navigate to Page 1
            </button>
        </ion-content>
    </ion-view>
</script>
```

注意看我们移除了*ion-view*上面的所有属性和指令然后给他添加了*ng-controller*。
我们需要去*www/js/app.js*里面更新这两个生命的控制器：

```
.controller('PageOneCtrl', function($scope, $ionicNavBarDelegate)
{
    $ionicNavBarDelegate.title('Page 1');
})
.controller('PageTwoCtrl', function($scope, $ionicNavBarDelegate)
{
    $ionicNavBarDelegate.title('Page 2');
    $ionicNavBarDelegate.showBackButton(false);
})
```

我们给*page1*和*page2*设置了标题,并且将*page2*的*showBackButton*设置为*false*。保存这些修改返回浏览器的时候，就可以看到你预想的结果了。
其他可以通过*$ionicNavBarDelegate*控制的属性有：

- align：这个是用来指定标题，按钮的排列方向的:left, right, 以及center（默认）
- showBar：用来设置或者取得*ion-nav-bar*是否显示

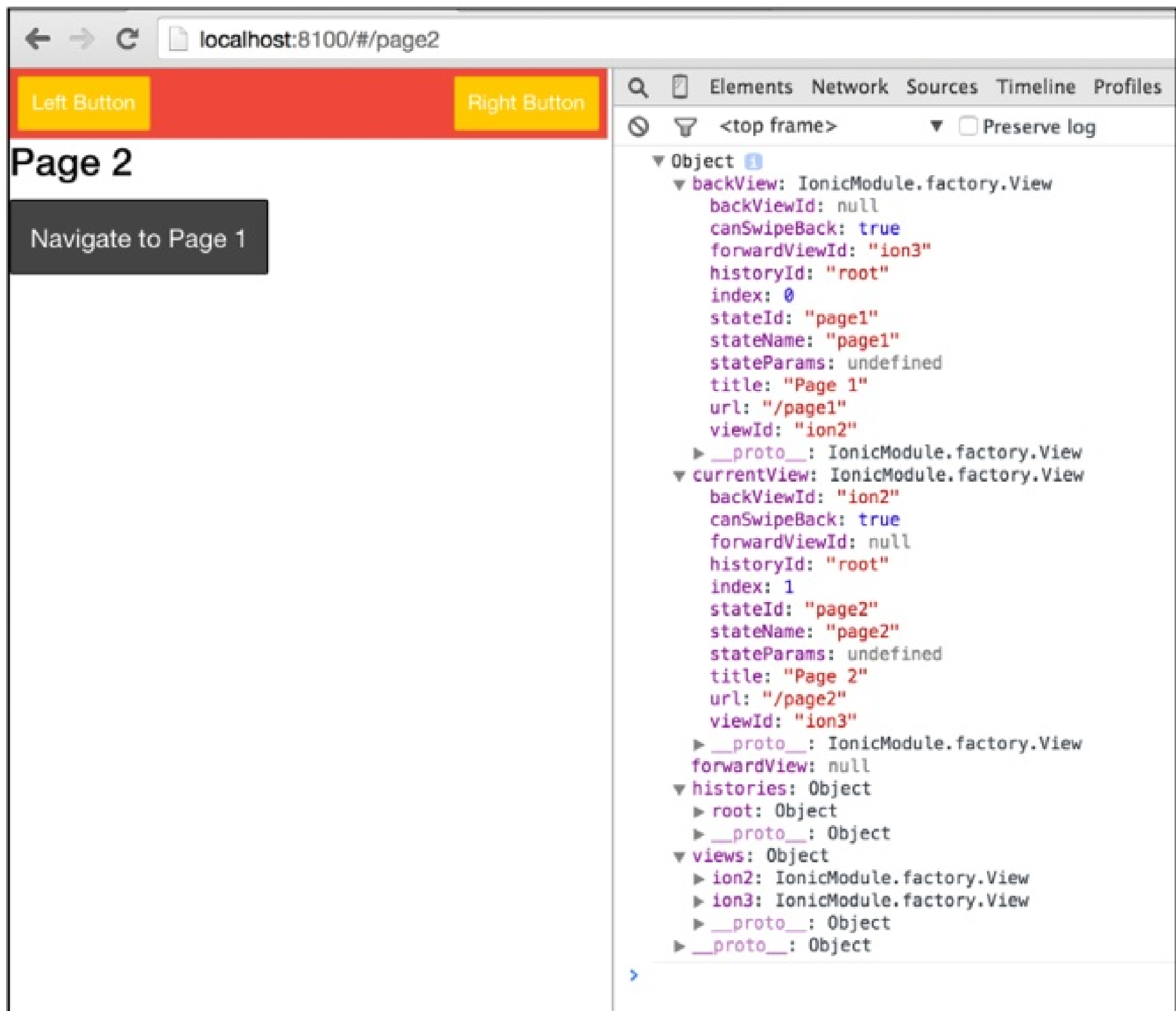
$ionicHistory

另一个非常重要的导航服务是*$ionicHistory*。*$ionicHistory*持续追踪所有视图并记录用户在视图之间的导航行为。
*$ionicHistory*的优美之处在于他支持平行历史记录，在这点上，浏览器是按固定顺序存储的。

当你的界面有多个标签，每个标签都有一套视图的时候，平行历史记录就会非常有用了。*$ionicHistory*可以用来捕获标签级别的历史记录；意思是，当用户选择标签2的时候，在标签2里面进行了一些页面导航，之后选择标签1，然后点击返回按钮；应用不会把用户带回标签2最后显示的页面， 而是导航到标签页的父容器的最后显示页，或者如果当前父容器页面的第一个页面的话就退出了应用。
回到手边的范例，更新*PageTwoCtrl*为以下：

```
.controller('PageTwoCtrl', function($scope, $ionicNavBarDelegate, $ionicHistory) {
    $ionicNavBarDelegate.title('Page 2');
    $ionicNavBarDelegate.showBackButton(false);
    console.log($ionicHistory.viewHistory())
})
```

我们在*PageTwoCtrl*中注入*$ionicHistory*依赖然后在控制台记录视图切换历史记录。
保存文件，返回浏览器，然后从page1导航到page2，我们可以看到如下展示：



*viewHistory*方法返回了一个对象，里面有*backView*（也就是之前视图），*currentView*，*Histtories*以及应用里面所有其他的视图的所有信息。
可以通过这个对象得到用户是如何导航到当前页面的；在此情景之下，视图历史非常有用。

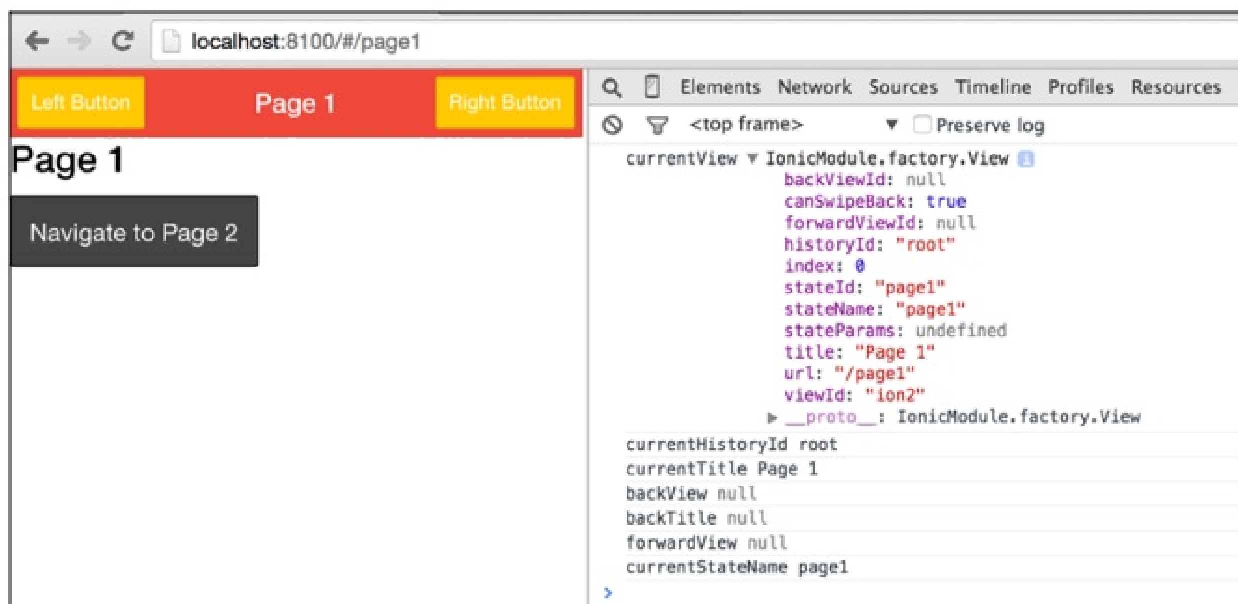
你依然可以通过*$ionicHistory*服务的函数读取视图历史的独立属性，例如：

- currentView：返回应用当前页面
- currentHistoryId：返回当前视图父容器的ID
- currentTitle：取得或者设置当前视图的标题
- backView：返回当前视图在历史记录栈里面的上一个视图
- forwardView：然后历史栈中的下一个视图。前视图当中用户从page1导航至page2，然后返回page1的时候有效。此时page2就是*forwardView*。
- currentStateName：返回当前状态名

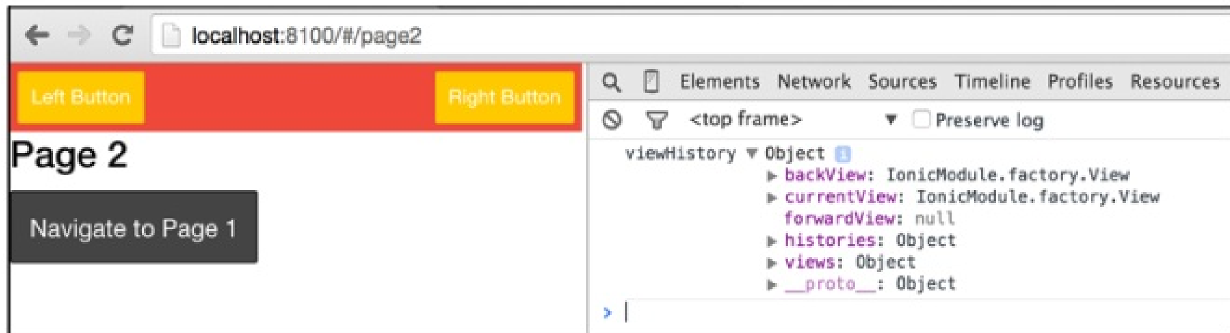
为快速测试以上属性，我们更新*PageOneCtrl*和*PageTwoCtrl*如下：

```
.controller('PageOneCtrl', function($scope, $ionicNavBarDelegate,$ionicHistory) {
    $ionicNavBarDelegate.title('Page 1');
    console.log('currentView', $ionicHistory.currentView());
    console.log('currentHistoryId', $ionicHistory.currentHistoryId());
    console.log('currentTitle', $ionicHistory.currentTitle());
    console.log('backView', $ionicHistory.backView());
    console.log('backTitle', $ionicHistory.backTitle());
    console.log('forwardView', $ionicHistory.forwardView());
    console.log('currentStateName', $ionicHistory.currentStateName());
})
.controller('PageTwoCtrl', function($scope, $ionicNavBarDelegate,$ionicHistory) {
    $ionicNavBarDelegate.title('Page 2');
    $ionicNavBarDelegate.showBackButton(false);
    console.log('viewHistory', $ionicHistory.viewHistory());
})
```

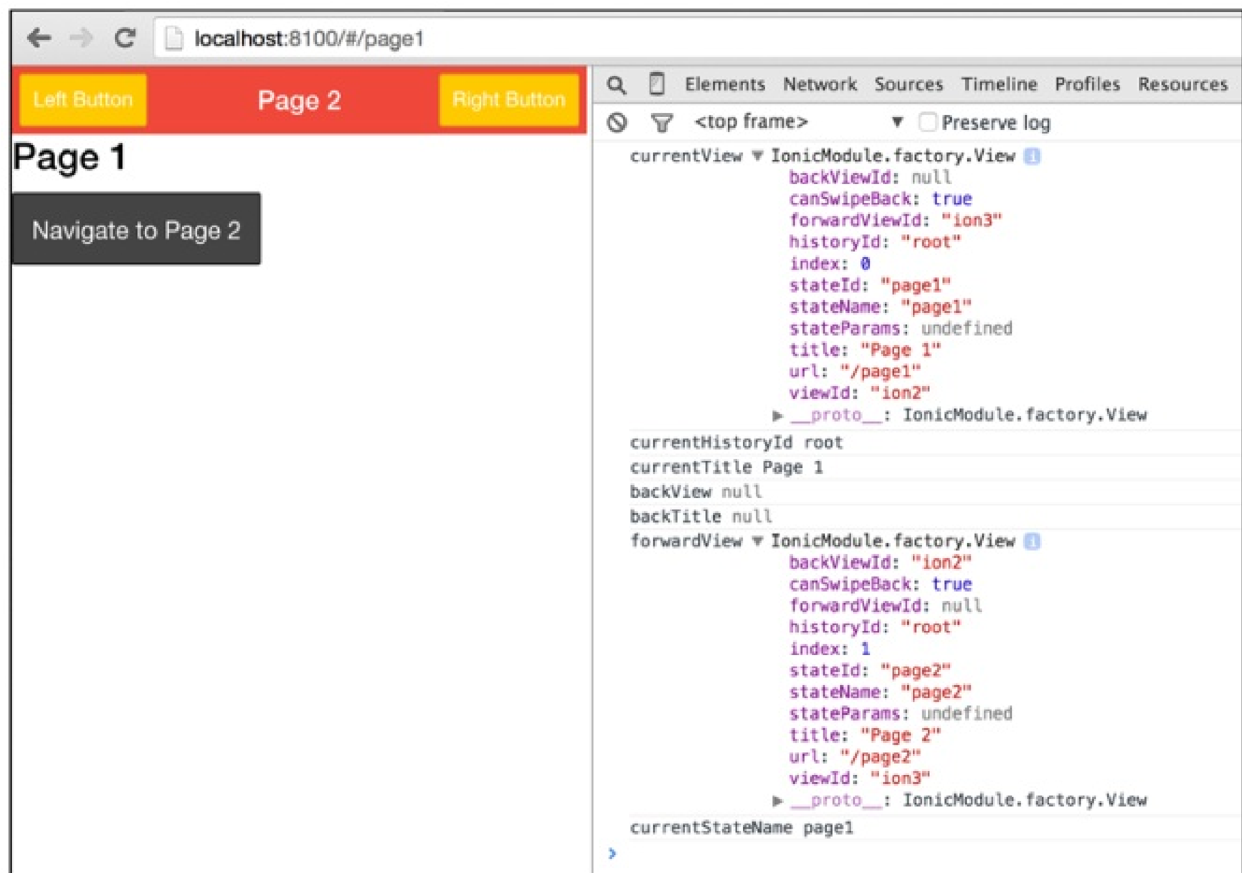
现在，当我们导航到page1的时候，可以记录的属性如下：



然后，当导航到**Page 2**的时候，可以看到早先看到的相同的值：



最后，当你再次导航回**Page 1**的时候，将会看到有*forwardView*了：



> 我已经为page1和page的*ion-view*指令设置了 *cache-view="false"*；因此，上面截屏中的*backView*是*null*。

*$ionicHistory*还有3个其他的方法：

- goBack：默认状态路由返回1个视图的历史距离。你可以传入一个负整数来制定返回多少个视图的历史距离。因为默认值是-1，所以返回的是一个视图的历史距离。如果执行*goBack(-2)*那么将返回两个视图的历史距离。 *goBack*不会超过历史栈，如果传入的值超过历史栈的话，会直接返回到第一个页面。
- clearHistory：清除除了当前页面之外的其他历史栈
- clearCache：清除所有缓存的视图

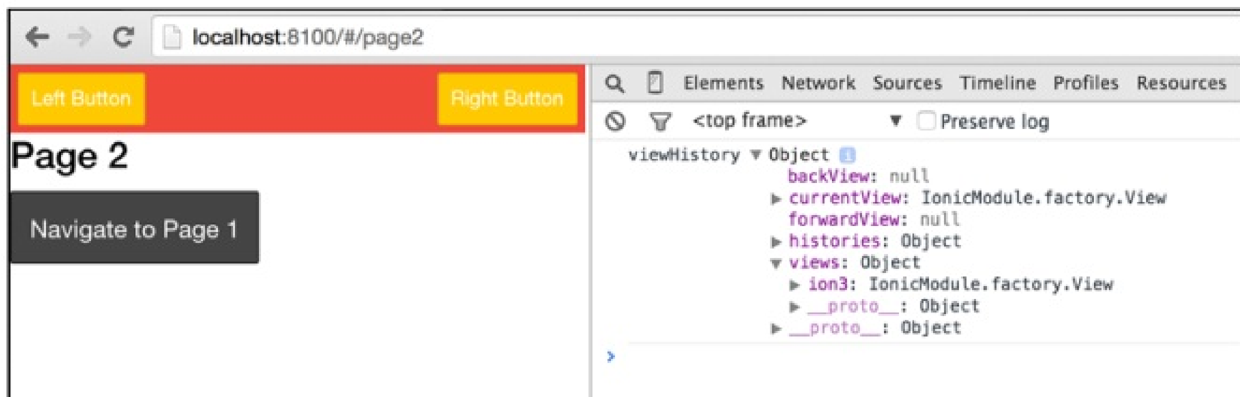
可以使用*$ionicHistory*的*nextViewOptions*方法来控制下一个视图如何展示。
你可以控制以下选项：

- disableAnimate：禁用下一个视图的动画效果
- disableBack：将下一个视图的*backView*设为*null*
- historyRoot：将下一个视图设置为历史栈的根

将以上属性添加到我们的*PageOneCtrl*：

```
.controller('PageOneCtrl', function($scope, $ionicNavBarDelegate,$ionicHistory) {
    $ionicNavBarDelegate.title('Page 1');
    console.log('currentView', $ionicHistory.currentView());
    console.log('currentHistoryId', $ionicHistory.currentHistoryId());
    console.log('currentTitle', $ionicHistory.currentTitle());
    console.log('backView', $ionicHistory.backView());
    console.log('backTitle', $ionicHistory.backTitle());
    console.log('forwardView', $ionicHistory.forwardView());
    console.log('currentStateName', $ionicHistory.currentStateName());
    $ionicHistory.nextViewOptions({
        disableAnimate: true,
        disableBack: true,
        historyRoot: true
    });
})
```

此时，导航到page2，控制台的输出如下：



你会发现，当点击**Navigate to Page 2**的时候，动画效果和过度效果都被禁用了，*backView*是*null*，最后*views*属性只有一个视图：*page2*。
你也可以使用这些选项来控制你的应用的历史状态的行为。

标签与侧边菜单

为更好的理解导航，我们将探索一下*tabs*指令与*side menu*指令。
新建一个*tabs*模板项目，后面学习一下与标签相关的指令：

```
ionic start -a "Example 19" -i app.example.nineteen example19 tabs
```

使用*cd*命令进入*example19*文件夹，运行如下命令：

```
ionic serve
```

此时，标签页应用就运行起来了。
当你打开*www/index.html*的时候，你会发现这个模板是通过一个内置*ion-nav-back-button*的*ion-nav-bar*来管理页头的。
接着，打开*www/js/app.js*，找到应用的状态配置：

```
.state('tab.dash', {
    url: '/dash',
    views: {
        'tab-dash': {
            templateUrl: 'templates/tab-dash.html',
            controller: 'DashCtrl'
        }
    }
})
```

注意看*views*对象中有一个名为*tab-dash*的对象。这个会在使用*tabs*指令的时候用到。在选中标签页的时候，这个名字用来加载一个指定的名为*tab-dash*的视图到*ion-nav-view*指令中。
打开*www/templates/tabs.html*，你将会发现标签页组件的html标记代码：

```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">
    <!-- Dashboard Tab -->
    <ion-tab title="Status" icon-off="ion-ios-pulse" icon-on="ionios-pulse-strong" hre
f="#/tab/dash">
        <ion-nav-view name="tab-dash"></ion-nav-view>
    </ion-tab>
    <!-- Chats Tab -->
    <ion-tab title="Chats" icon-off="ion-ios-chatboxes-outline"   icon-on="ion-ios-cha
tboxes" href="#/tab/chats">
        <ion-nav-view name="tab-chats"></ion-nav-view>
    </ion-tab>
    <!-- Account Tab -->
    <ion-tab title="Account" icon-off="ion-ios-gear-outline" icon-on="ion-ios-gear" hr
ef="#/tab/account">
        <ion-nav-view name="tab-account"></ion-nav-view>
    </ion-tab>
</ion-tabs>
```

由于标签状态是作为一个抽象路由的存在，*tabs.html*将会在其他子标签加载之前完成加载。*ion-tab*指令是嵌入在*ion-tabs*指令里面的，每个*ion-tab*指令都有一个*ion-nav-view*嵌入其中。当选中一个标签的时候，与*ion-nav-view*上的*name*属性同名路由将会在对应的标签页中进行加载。

非常整洁的架构！

> 更多关于*tabs*指令和他的服务的信息请参考：http://ionicframework.com/docs/nightly/api/directive/ionTabs/

接下来，我们将创建一个侧边菜单模板app，然后学习其中的导航：

```
ionic start -a "Example 20" -i app.example.twenty example20 sidemenu
```

通过*cd*命令，进入*example20*文件夹运行如下命令：

```
ionic serve
```

此时，侧边菜单app启动完成。

我们先从*www/index.html*开始。这个文件的body里面只有一个*ion-nav-view*。接下来，我们打开*www/js/app.js*。在这里，路由都按期望的定义好了。 但是注意观察search，browse和playlist的views的名字，他们都是一样的--*menuContent*：

```
.state('app.search', {
    url: "/search",
    views: {
        'menuContent': {
            templateUrl: "templates/search.html"
        }
    }
})
```

打开*www/templates/menu.html*，将会看到*ion-side-menu*指令。他有两个子元素，*ion-side-menu-content*和*ion-side-menu*。 *ion-side-menu-content*展示了*ion-nav-view*里面名为*menuContent*里的每个菜单条目。这就是为什么所有状态路由器里面的菜单条目名字一样的原因。

*ion-side-menu*显示在页面的左边。你可以设置他的位置在右边，也可以设置为两边都有。注意观察*ion-nav-buttons*内部的按钮上的*menu-toggle*指令。这个指令用来切换侧边菜单的显示。

如果想要两边都了菜单的话，*menu.html*看起来是如下效果：

```
<ion-side-menus enable-menu-with-back-views="false">
  <ion-side-menu-content>
    <ion-nav-bar class="bar-stable">
      <ion-nav-back-button>
      </ion-nav-back-button>
      <ion-nav-buttons side="left">
        <button class="button button-icon button-clear ionnavicon" menu-toggle="left">
    </button>
      </ion-nav-buttons>
      <ion-nav-buttons side="right">
        <button class="button button-icon button-clear ionnavicon" menu-toggle="right"
>
    </button>
      </ion-nav-buttons>
    </ion-nav-bar>
    <ion-nav-view name="menuContent"></ion-nav-view>
  </ion-side-menu-content>
  <ion-side-menu side="left">
    <ion-header-bar class="bar-stable">
      <h1 class="title">Left</h1>
    </ion-header-bar>
    <ion-content>
      <ion-list>
        <ion-item menu-close ng-click="login()">
          Login
        </ion-item>
        <ion-item menu-close href="#/app/search">
          Search
        </ion-item>
        <ion-item menu-close href="#/app/browse">
          Browse
```

```
        </ion-item>
        <ion-item menu-close href="#/app/playlists">
          Playlists
        </ion-item>
      </ion-list>
    </ion-content>
  </ion-side-menu>
  <ion-side-menu side="right">
    <ion-header-bar class="bar-stable">
      <h1 class="title">Right</h1>
    </ion-header-bar>
    <ion-content>
      <ion-list>
        <ion-item menu-close ng-click="login()">
          Login
        </ion-item>
        <ion-item menu-close href="#/app/search">
          Search
        </ion-item>
        <ion-item menu-close href="#/app/browse">
          Browse
        </ion-item>
        <ion-item menu-close href="#/app/playlists">
          Playlists
        </ion-item>
      </ion-list>
    </ion-content>
  </ion-side-menu>
</ion-side-menus>
```

> 参考此处查看更多关于*side menu*指令和他的服务信息：
> http://ionicframework.com/docs/nightly/api/directive/ionSideMenus

Ionic加载

第一个学习的服务是*$ionicLoading*。这个服务在你想要从主页上阻断用户交互的时候，或者告诉用户后台正在进行一些处理的时候非常有用。
新建一个空白模板项目并实现*$ionicLoading*来进行测试:

```
ionic start -a "Example 21" -i app.example.twentyone example21 blank
```

使用*cd*命令进入*example21*文件夹，运行：

```
ionic serve
```

然后这个项目将会运行在浏览器中。

然后我们创建一个应用控制器，在其中定义显示和隐藏的方法。 打开*www/js/app.js*添加以下代码：

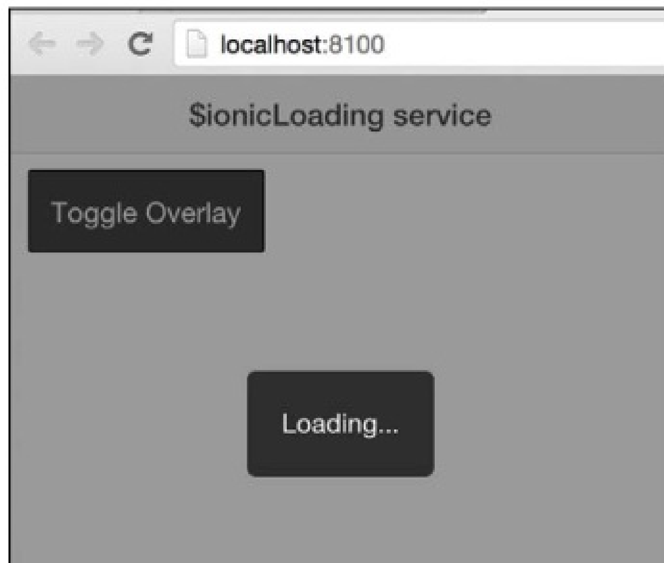
```
.controller('AppCtrl', function($scope, $ionicLoading, $timeout) {
    $scope.showLoadingOverlay = function() {
        $ionicLoading.show({
            template: 'Loading...'
        });
    };
    $scope.hideLoadingOverlay = function() {
     $ionicLoading.hide();
    };
    $scope.toggleOverlay = function() {
        $scope.showLoadingOverlay();
        // wait for 3 seconds and hide the overlay
        $timeout(function() {
            $scope.hideLoadingOverlay();
        }, 3000);
    };
})
```

我们有一个叫做*showLoadingOverlay*的方法，这个方法将调用*$ionicLoading.show()*，还有一个叫做*hideLoadingOverlay*的方法，这个方法将调用*$ionicLoading.hide()*。 同时我们也创建了一个名为*toggleOverlay()*的工具方法，这个方法将会调用*showLoadingOverlay*方法，3秒钟之后调用*hideLoadingOverlay*。

我们将在*www/index.html*的body部分更新如下显示：

```
<body ng-app="starter" ng-controller="AppCtrl">
    <ion-header-bar class="bar-stable">
    <h1 class="title">$ionicLoading service</h1>
    </ion-header-bar>
    <ion-content class="padding">
    <button class="button button-dark" ng-click="toggleOverlay()">
    Toggle Overlay
    </button>
    </ion-content>
</body>
```

我们有一个按钮调用*toggleOverlay*。
保存所有文件，回到浏览器，点击**Toggle Overlay**按钮，将会看到如图效果：



覆盖层将会在*$ionicLoading*调用*hide*方法之前一直显示。
你可以将上面的逻辑放入一个服务中，然后在应用中重复利用。服务代码如下：

```
.service('Loading', function($ionicLoading, $timeout) {
    this.show = function() {
        $ionicLoading.show({
            template: 'Loading...'
        });
    };
    this.hide = function() {
        $ionicLoading.hide();
    };
    this.toggle= function() {
        var self = this;
        self.show();
        // wait for 3 seconds and hide the overlay
        $timeout(function() {
            self.hide();
        }, 3000);
    };
})
```

现在，当你想在你的控制器或者指令中注入了*Loading*服务，你就可以使用*Loading.show()*,*Loading.hide()*和*Loading.toggle()*。
如果你只是想展示一个微调图标而不是文本的话，可以直接调用*$ionicLoading.show*方法，不使用任何选项：

```
$scope.showLoadingOverlay = function() {
    $ionicLoading.show();
};
```

然后，你就可以看到下面这样的效果：



> 你可以后续再配置*show*方法。更新信息参考：
> http://ionicframework.com/docs/nightly/api/service/$ionicLoading/ 可以使用*$ionicBackdrop*服务来展示一个背景。
> http://ionicframework.com/docs/nightly/api/service/$ionicBackdrop *$ionicModal*与加载服务差不多： http://ionicframework.com/docs/api/service/$ionicModal

动作表单服务 Action Sheet service

动作表单是一个向上滑动的展示了一系列选项的面板。通常，当你有一组列表条目或者格子条目的时候，他用来显示上下文选项。动作表单服务一般用在用户长按列表或者栅格条目的时候。
为测试*$ionicActionSheet*服务，我们新建一个空白模板项目：

```
ionic start -a "Example 22" -i app.example.twentytwo example22 blank
```

使用*cd*口令，进入*example22*文件夹然后运行如下服务：

```
ionic serve
```

然后浏览器中就启动了这个项目了。
打开*www/js/app.js*新建一个控制器，叫做*AppCtrl*：

```
.controller('AppCtrl', function($scope, $ionicActionSheet,$timeout) {
    $scope.showOptions = function() {
        var hideSheet = $ionicActionSheet.show({
            buttons: [{
                text: 'Open'
            }, {
                text: 'Get Link'
            }],
            destructiveText: 'Delete',
            titleText: 'Options'
        });
        // hide the sheet after three seconds
        $timeout(function() {
            hideSheet();
        }, 3000);
    };
})
```

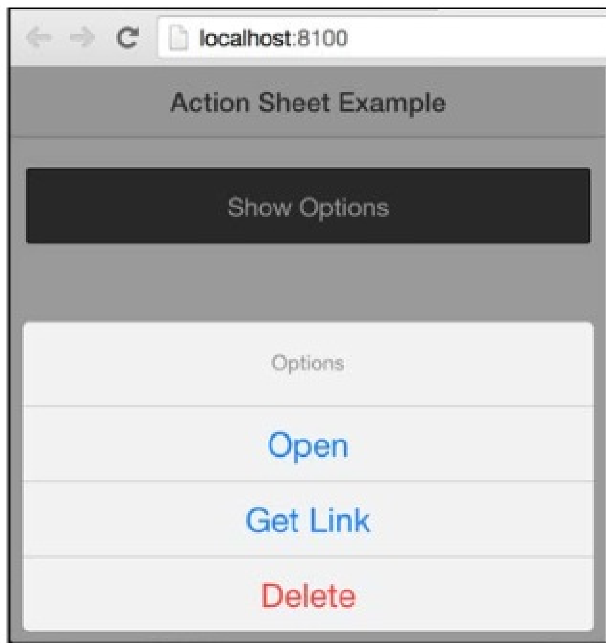
*$ionicActionSheet.show*返回了一个方法，当这个方法执行的时候，关闭了动作表单。*show*方法接受一个对象作为参数，这个对象有以下几个属性：

- buttons：这个显示了一个选项或者按钮列表。
- destructiveText：高亮一个指定的选项作为一个危险操作
- titleText：设置动作表单的标题

然后更新*www/index.html*的body部分如下：

```
<body ng-app="starter" ng-controller="AppCtrl">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Action Sheet Example</h1>
        </ion-header-bar>
        <ion-content class="padding">
            <button class="button button-block button-dark" ng-click="showOptions()">
            Show Options
            </button>
        </ion-content>
    </ion-pane>
</body>
```

保存所有文件，回到浏览器，然后会看到一个**Show Options**按钮。点击将会看到如下效果：



这个动作表单将会在3秒后隐藏。

> 在第八章 制作一个聊天*app*的实际操作中，我们将会用到动作表单；其中我们会实现动作表单的按钮处理器。 更多关于动作表单的信息参考：
> http://ionicframework.com/docs/nightly/api/service/$ionicActionSheet/

Popover与Popup服务

Popover通常是显示在选中条目旁边的一个上下文视图。这个组件用来显示上下文信息或者显示某组件的更多信息。
新建一个空白模板项目来进行学习：

```
ionic start -a "Example 23" -i app.example.twentythree example23 blank
```

使用*cd*口令进入*example23*，运行：

```
ionic serve
```

浏览器中将运行此项目。
给项目添加一个新的控制器，名为*AppCtrl*。控制器代码是添加在*www/js/app.js*中的：

```
.controller('AppCtrl', function($scope, $ionicPopover) {
    // init the popover
    $ionicPopover.fromTemplateUrl('button-options.html', {
        scope: $scope
    }).then(function(popover) {
        $scope.popover = popover;
    });
    $scope.openPopover = function($event, type) {
        $scope.type = type;
        $scope.popover.show($event);
    };
    $scope.closePopover = function() {
        $scope.popover.hide();
        // if you are navigating away from the page once
        // an option is selected, make sure to call
        // $scope.popover.remove();
    };
});
```

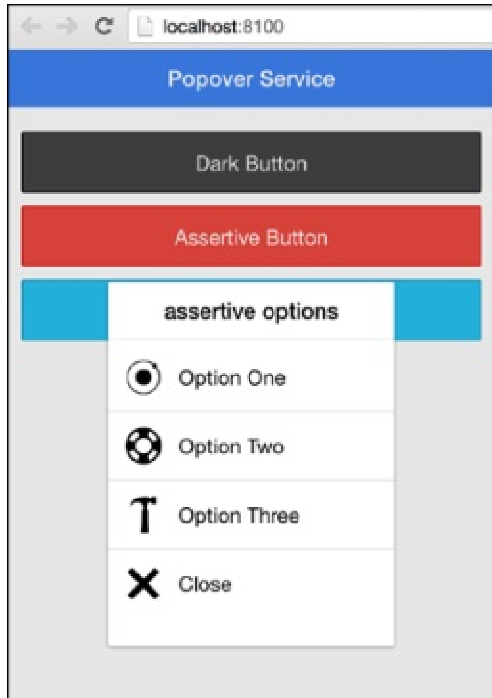
我们使用的是*$ionicPopover*服务，同一个一个名为*buttons-options.html*的模板设置popover的。可以将当前控制器的scope作为scope传给popover。 控制器scope上有两个方法用来显示和隐藏popover。*openPopover*方法接受两个选项。一个是事件，另一个是我们当前点击的按钮的类型（同时的）。

接下来，将*www/index.html*的body部分改为如下：

```
<body ng-app="starter" ng-controller="AppCtrl">
  <ion-header-bar class="bar-positive">
    <h1 class="title">Popover Service</h1>
  </ion-header-bar>
  <ion-content class="padding">
    <button class="button button-block button-dark" ngclick="openPopover($event, 'dark
')">
    Dark Button
    </button>
    <button class="button button-block button-assertive" ng-click="openPopover($event,
 'assertive')">
    Assertive Button
    </button>
    <button class="button button-block button-calm" ng-click="openPopover($event, 'cal
m')">
    Calm Button
    </button>
  </ion-content>
  <script id="button-options.html" type="text/ng-template">
    <ion-popover-view>
      <ion-header-bar>
        <h1 class="title">{{type}} options</h1>
      </ion-header-bar>
      <ion-content>
        <div class="list">
          <a href="#" class="item item-icon-left">
            <i class="icon ion-ionic"></i> Option One
          </a>
          <a href="#" class="item item-icon-left">
            <i class="icon ion-help-buoy"></i> Option Two
          </a>
          <a href="#" class="item item-icon-left">
            <i class="icon ion-hammer"></i> Option Three
          </a>
          <a href="#" class="item item-icon-left" ng-click="closePopover()">
            <i class="icon ion-close"></i> Close
          </a>
        </div>
      </ion-content>
    </ion-popover-view>
  </script>
</body>
```

在*ion-content*中，我们创建了3个按钮，每个的心情颜色都不一样（黑暗，武断与冷静）。当用户点击按钮的时候，显示按钮指定的popover。 在这个范例中，我们只是把心情传进去，然后将心情作为popover的页头。明显，你可以做更多的逻辑。

注意，我们的模板内容都是包装在*ion-popover-view*里面的。他会负责对恰当的modal对位。

> 为了使popover工作正常，模板必须包装在*ion-popover-view*里面。 保存所有文件，返回浏览器，我们会看到3个按钮。点击其中一个按钮，popover的页头将会改变，但是选项却都是一样的:
>
> 
>

然后，当我们点击页面上的任何地方或者关闭选项的时候，popover就会关闭。

> 如果在选中选项的时候想要导航到其他页面的话，一定要调用:**$scope.popover.remove();** 更多关于Popover的信息，参考：
> http://ionicframework.com/docs/api/controller/ionicPopover/

> 词汇求助：
>
> - popover:
> - pin dialog:

ionicPopup

接下来学习的服务是*$ionicPopup*。这个服务用来创建一个弹出框以供用户响应来确定是否继续。
这些弹出框都是JavaScript本身的*alert*，*prompt*和*confirm*方法的自定义样式。
新建一个空白模板项目进行测试：

```
ionic start -a "Example 24" -i app.example.twentyfour example24 blank
```

通过*cd*命令进入到*example24*文件夹运行如下命令：

```
ionic serve
```

然后浏览器中将会运行此项目。
我们将会实现展示，确认和警告方法的app样式。
我们将使用*show*方法显示一个pin对话框，用户需要在这个对话框中输入pin。如果pin有效，我们就给用户展示一个我们代码的安全区域。 安全区域有一些展示确认框和警告框的按钮。如果用户直接退出了pin对话框，我们将把用户带到一个不安全区域，然后重新询问用户一遍。
此范例使用AngularJS和Ionic介绍了一个根据条件显示内容的途径。
创建一个*AppCtrl*控制器作为开始。在*www/js/app.js*中，添加以下代码：

```
.controller('AppCtrl', function($scope, $ionicPopup) {
    $scope.data = {};
    $scope.state = {};
    $scope.error = {};
    $scope.prompt = function() {
        // reset app states
        $scope.state.cancel = false;
        $scope.state.success = false;
        // reset error messages
        $scope.error.empty = false;
        $scope.error.invalid = false;
        var prompt = $ionicPopup.show({
            templateUrl: 'pin-template.html',
            title: 'Enter Pin to continue',
            scope: $scope,
            buttons: [{
                text: 'Cancel',
```

```
                onTap: function(e) {
                    $scope.state.cancel = true;
                }
            }, {
            text: '<b>Login</b>',
            type: 'button-assertive',
            onTap: function(e) {
                $scope.error.empty = false;
                $scope.error.invalid = false;
                if (!$scope.data.pin) {
                    // disable close if the
                    // user does not enter
                    // a valid pin
                    $scope.error.empty = true;
                    e.preventDefault();
                } else {
                    if ($scope.data.pin === '1234') {
                        $scope.state.success = true;
                        return $scope.data.pin;
                    } else {
                        $scope.error.invalid = true;
                        e.preventDefault();
                    }
                }
            }
            }]
        });
    };
    $scope.confirm = function() {
        var confirm = $ionicPopup.confirm({
            title: 'Confirm Popup Heading',
            template: 'Are you sure you want to do that?'
        });
        confirm.then(function(res) {
            if (res) {
                console.log('Yes!');
            } else {
                console.log('Nooooo!!');
            }
        });
    };
    $scope.alert = function() {
        var alert = $ionicPopup.alert({
            title: 'You are secured!',
            template: 'You are inside a secure area!'
        });
        alert.then(function(res) {
            console.log('Yeah!! I know!!');
        });
    };
    // invoke the prompt on controller init.
    $scope.prompt();
})
```

代码好多啊！！（作者原话）但是也很简单。（作者原话）好长，懒得看，后续跑代码的时候再看。（译者原话）

我们在scope上创建了3个对象，分别名为*data*，*state*，和*error*。这些对象将用来存储数据，应用状态和错误信息。

我们添加了一个*prompt*方法。在这个方法里面，我们调用了*$ionicPopup.show*方法，传入模板和取消按钮以及**Login**按钮的点击方法的定义。 当用户点击了prompt的**Cancel**按钮的时候，我们将状态对象的*cancel*属性设置为*true*。这个属性将用作切换视图。

当用户点击**Login**按钮的时候，我们将会检查是否输入了有效的pin。如果没有的话，我们会将对于的错误信息设置为*true*。 如果用户输入了有效的pin并且等于*1234*的话，我们会把状态对象上的success属性设置为*true*。这样将会切换到另一个视图，这个视图上将会有**Confirm**和**Alert**按钮。

*confirm*和*alert*方法都是自解释性的。他们分别使用*$ionicPopup.confirm*和*$ionicPopup.alert*进行设置。 这些方法返回一个promise，当在点击按钮的时候，就可以解析了。

最后，我们在控制器加载的时候调用*prompt*方法来显示*Pin*对话框。

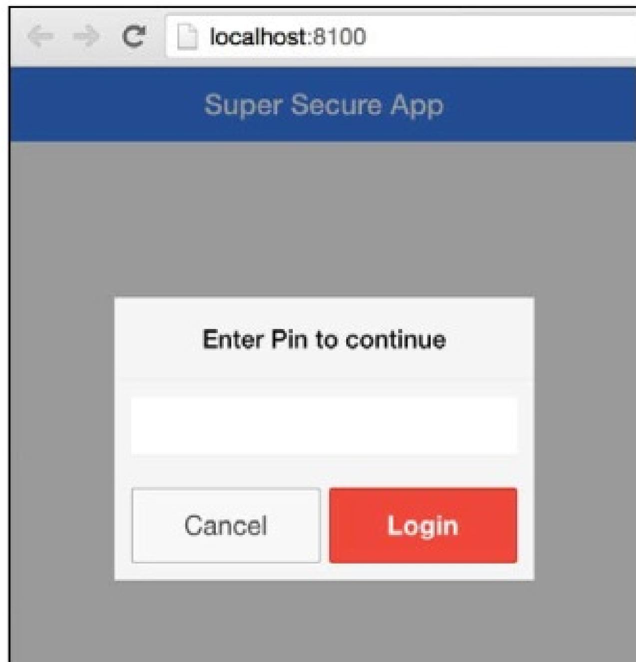
*www/index.html*的body部分更新如下：

```
<body ng-app="starter" ng-controller="AppCtrl">
<ion-pane ng-cloak>
    <ion-header-bar class="bar-positive">
        <h1 class="title">Super Secure App</h1>
    </ion-header-bar>
<ion-content class="padding">
    <div class="card" ng-show="state.cancel">
        <div class="item item-divider">
        Oops!! you cancelled!
        </div>
        <div class="item item-text-wrap">
        To see the secure content enter pin
            <button class="button button-assertive buttonblock" ng-click="prompt()">
            Try Again!
            </button>
        </div>
    </div>
    <div class="card" ng-show="state.success">
        <div class="item item-divider">
        You are viewing secure content!
        </div>
        <div class="item item-text-wrap">
            <button class="button button-positive buttonblock" ng-click="confirm()">
            Show Confirm Dialog
            </button>
            <button class="button button-positive buttonblock" ng-click="alert()">
            Show Alert Dialog
            </button>
        </div>
    </div>
    </ion-content>
</ion-pane>
<script type="text/ng-template" id="pin-template.html">
    <input type="password" ng-model="data.pin">
    <label ng-show="error.empty" class="assertive text-center
    block padding">Please enter a valid Pin</label>
    <label ng-show="error.invalid" class="assertive textcenter block padding">Invalid
Pin, Try Again!</label>
</script>
</body>
```

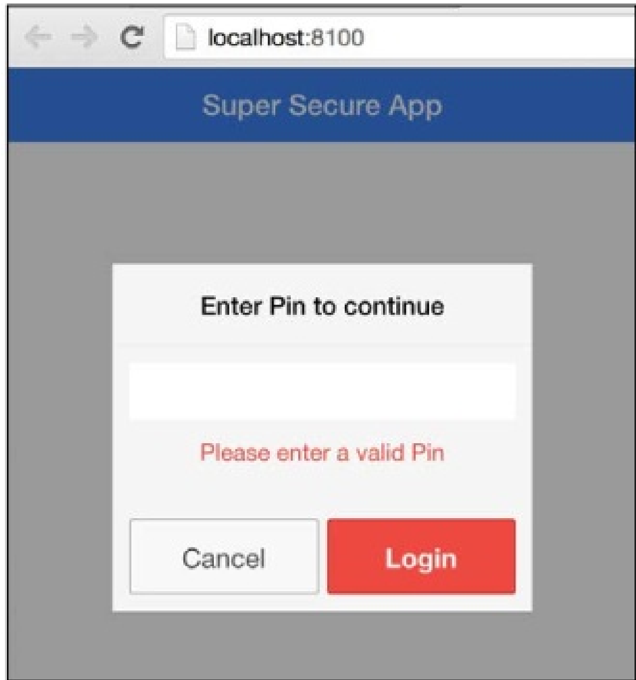
我们添加了两个卡片视图；一个在*state.cancel*为*true*的时候显示，另一个在*state.success*为*true*的时候显示。在body标签的最后，我们添加了*pin-template.html*模板。
重点关注一下我们给*ion-panel*指令添加的*ng-cloak*属性。这个属性确保了AngularJS处理完成之前不显示任何内容。

> 更多信息关于*ng-cloak*请参考：https://docs.AngularJS.org/api/ng/directive/ngCloak

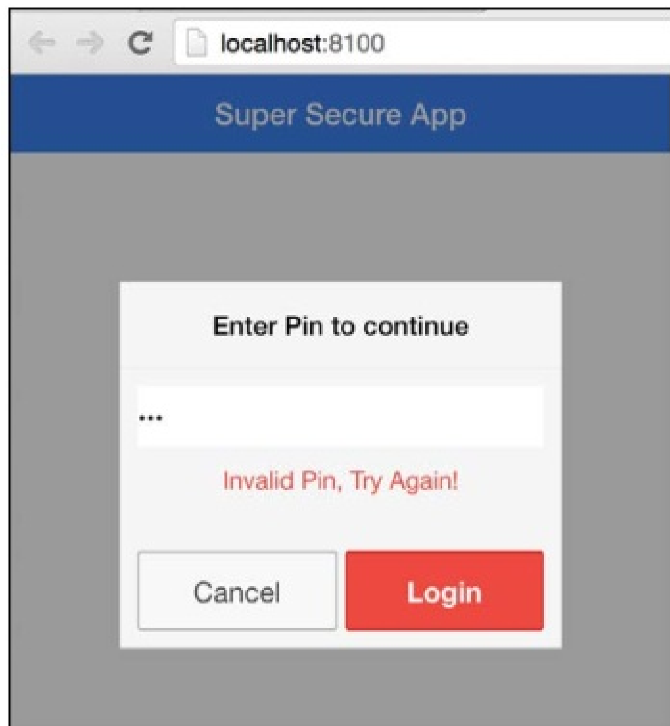
保存所有文件，返回浏览器，将会看到：



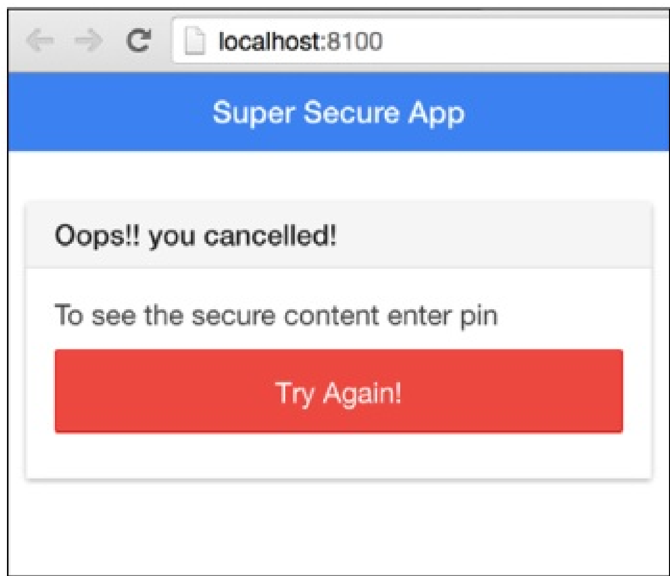
在不输入任何数据的情况下点击**Login**，将会显示：



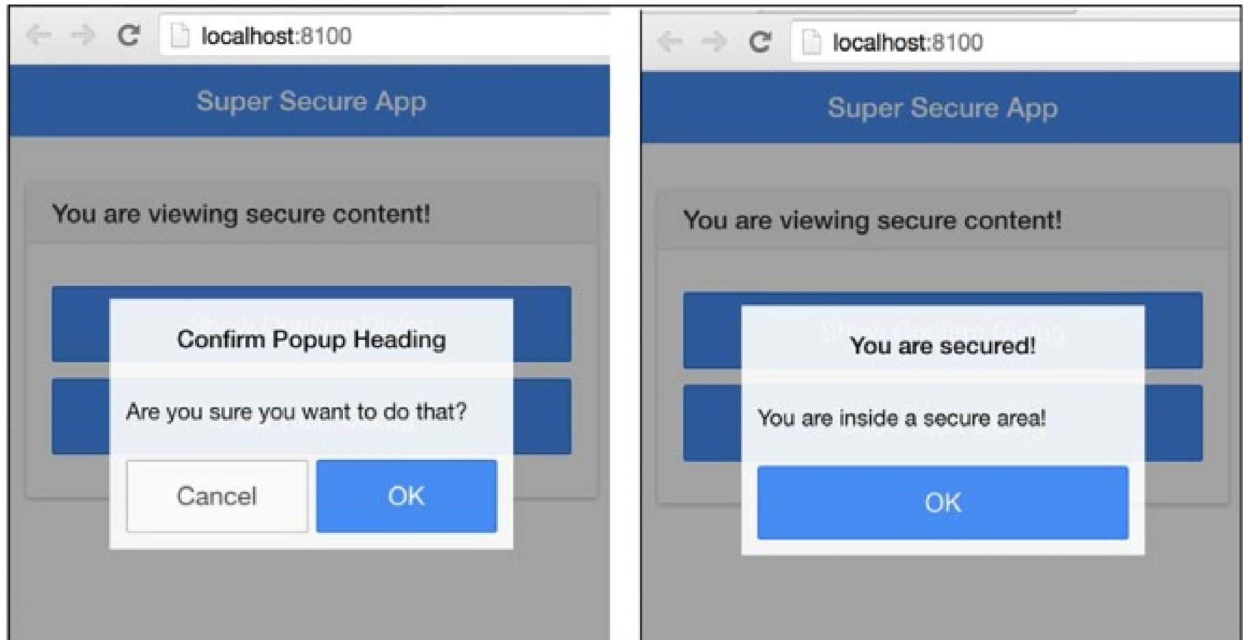
在输入一个无效的pin的时候，将会看到：



取消弹出框的时候，会把你带到一个不安全的区域，这里更你另一次做人的机会：



终于，在你输入了有效的pin值之后，你会被带到一个安全区域，其中可以看到**Confirm**和**Alert**按钮。点击他们之后会看到：



上面的代码不仅讨论了*$ionicPopup*服务，也让你知道了如何搭建自己的应用。

ion-list和ion-item指令

鉴于咱们是在熟悉大部分的Ionic指令和服务，我想应该值得提一下*ion-list*和*ion-item*指令。
在移动应用中，列表是使用最广泛的显示模式之一。在Ionic中，我们可以像在第三章 *Ionic CSS*组件和导航中那样使用CSS版本的列表，或者使用指令版的列表。
使用指令版的列表的好处是他提供了很多额外属性用来更好的管理列表，例如：*ion-delete-button*, *ion-reorder-button*以及*ion-option-button*。
新建一个空白模板项目进行测试:

```
ionic start -a "Example 25" -i app.example.twentyfive example25 blank
```

老规矩：

```
ionic serve
```

然后浏览器就运行了这个项目了。
我们将实现一个Ionic文档里面提供的范例，涵盖上面提到的所有指令。

> 接下来的范例来源于这里，不同的是我们改为通过工厂添加数据的：
> http://codepen.io/ionic/pen/JsHjf

首先，我们创建一个工厂用来分发随机的数据给列表。工厂和我们在*example17*里面使用的那个差不多，除了返回的数据有所不同：

```
.factory('DataFactory', function($timeout, $q) {
    var API = {
        getData: function(count) {
            // Spoof a network call using promises
            var deferred = $q.defer();
            var data = [],
            _o = {};
            count = count || 20;

            for (var i = 0; i < count; i++) {
                _o = {
                    // http://stackoverflow.com/a/8084248/1015046
                    id: i + 1,
                    title: (Math.random() + 1).toString(36).substring(7)
                };
                data.push(_o);
            };

            $timeout(function() {
                // success response!
                deferred.resolve(data);
            }, 1000);

            return deferred.promise;
        }
    };
    return API;
})
```

此处返回了一个带有*id*和*title*属性的对象。

接下来会在*www/js/app.js*创建一个名为*AppCtrl*的控制器。他将负责从工厂拿取数据以及建立列表。同时，我们将为**Edit**, **Delete**以及**Option**按钮提供点击处理函数：

```
.controller('AppCtrl', function($scope, DataFactory) {
    $scope.items = [];
    $scope.data = {
        showDelete: false
    };

    $scope.edit = function(item) {
        alert('Edit Item: ' + item.id);
    };

    $scope.share = function(item) {
        alert('Share Item: ' + item.id);
    };

    $scope.moveItem = function(item, fromIndex, toIndex) {
        $scope.items.splice(fromIndex, 1);
        $scope.items.splice(toIndex, 0, item);
    };

    $scope.onItemDelete = function(item) {
        $scope.items.splice($scope.items.indexOf(item), 1);
    };

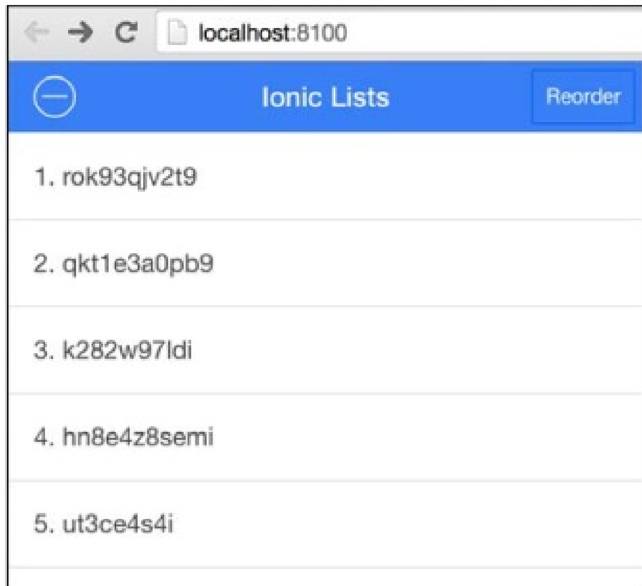
    // get data on page load
    DataFactory.getData().then(function(data) {
     $scope.items = data;
    });
})
```

接着，修改*www/index.html*的body部分：

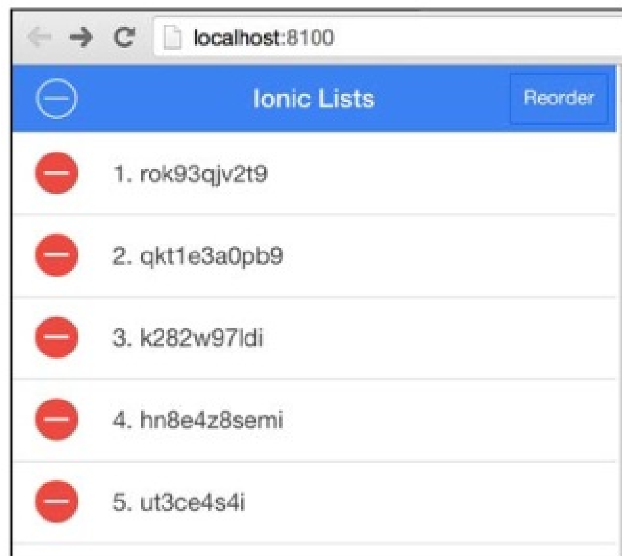
```
<body ng-app="starter" ng-controller="AppCtrl">
  <!-- http://codepen.io/ionic/pen/JsHjf -->
  <ion-header-bar class="bar-positive">
    <div class="buttons">
      <button class="button button-icon icon ion-ios-minusoutline" ng-click="data.show
Delete = !data.showDelete; data.showReorder = false"></button>
    </div>
    <h1 class="title">Ionic Lists</h1>
    <div class="buttons">
      <button class="button" ng-click="data.showDelete =
    false; data.showReorder = !data.showReorder">
        Reorder
        </button>
    </div>
  </ion-header-bar>
  <ion-content>
    <ion-list show-delete="data.showDelete" show-reorder="data.showReorder">
      <ion-item ng-repeat="item in items" item="item" class="item-remove-animate">
        {{ item.id }}. {{ item.title }}
        <ion-delete-button class="ion-minus-circled" ng-click="onItemDelete(item)">
        </ion-delete-button>
        <ion-option-button class="button-assertive" ng-click="edit(item)">
          Edit
        </ion-option-button>
        <ion-option-button class="button-calm" ng-click="share(item)">
          Share
        </ion-option-button>
        <ion-reorder-button class="ion-navicon" on-reorder="moveItem(item, $fromIndex,
 $toIndex)"></ion-reorderbutton>
      </ion-item>
    </ion-list>
  </ion-content>
</body>
```

页头里有两个两个按钮用来切换列表上的删除和重排图标。在*ion-list*指令中，我们使用了*show-delete*和*show-reorder*属性来显示和隐藏列表条目上的图标。
在每个*ion-item*指令中，我们添加了*ion-delete-button*来调用*onItemDelete*函数；添加了*ion-option-button*来显示**Share**和**Edit**按钮； 最后添加了*ion-reorfer-button*来显示重排图标。重排

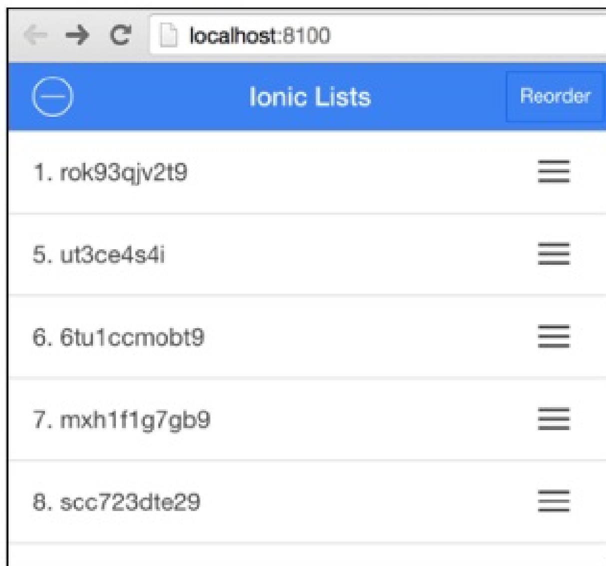
的时候，我们将调用*moveItem*方法。
保存所有文件，返回浏览器，可以看到：



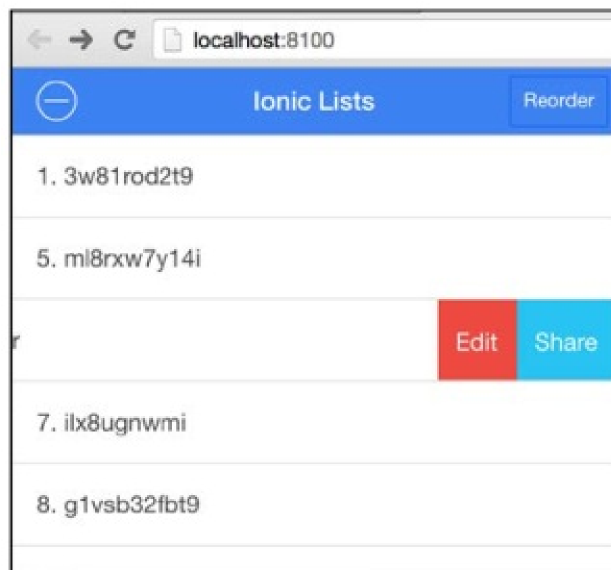
点击页头的删除图标的时候：



可以通过点击条目左边的删除图标来删除他。点击页头的重排按钮的时候：



可以使用每行提供的操作随便移着玩。你也可以关闭重排，擦掉条目的左边显区域，显



示**Share**和**Edit**选项。

> 在使用*collection-repeat*替代*ng-repeat*的情况下，重排将会有问题。 请参考：https://github.com/driftyco/ionic/issues/1714 更多列表信息，请参考：http://ionicframework.com/docs/api/directive/ionList/

手势指令和服务

接下来的指令和服务都是关于手势的。手势是用户在与应用交互是在屏幕上的操作行为。一个简单的例子就是，在屏幕上进行捏的手势的时候缩小，拉的时候进行放大。
Ionic通过*$ionicGesture*支持这些手势。
为了使讲解更通用，我将解释一个手势指令然后给你展示如何使用他。此逻辑可以应用与所有其他手势。
新建一个空白模板项目来测试*on-drag-up*指令：

```
ionic start -a "Example 26" -i app.example.twentysix example26 blank
```

使用*cd*口令进入*example26*文件夹运行如下命令：

```
ionic serve
```

之后，项目将会在浏览器中运行。
首先，在*www/js/app.js*中创建一个名为*AppCtrl*的控制器：

```
.controller('AppCtrl', function($scope, $ionicGesture) {
    $scope.scopeGesture = 'None';
    $scope.delegateGesture = 'None';

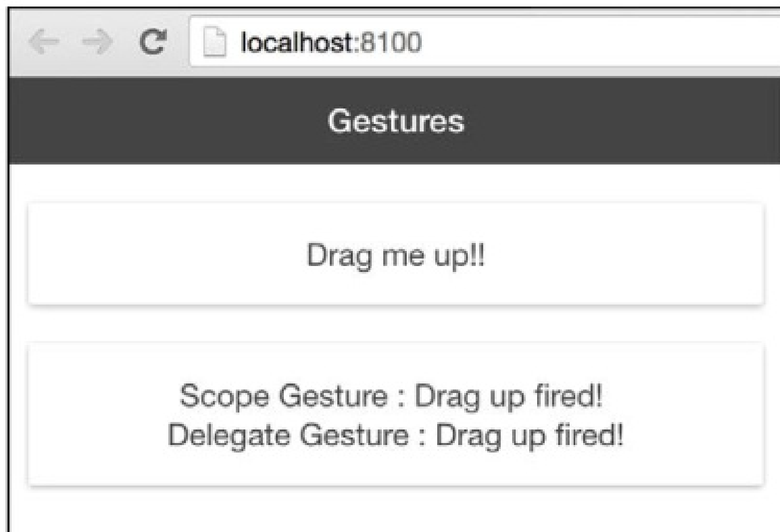
    $scope.onDragUp = function() {
        $scope.scopeGesture = 'Drag up fired!'
    };

    // Event listener using event delegation
    // The logic below would be typically written in a directive
    // We have added this to the controller for illustration
    purposes
    var $element =  angular.element(document.querySelector('#gestureContainer'));
    $ionicGesture.on('dragup', function() {
        $scope.delegateGesture = 'Drag up fired!';
    }, $element);
})
```

我们之前说过，我们要实现*dragup*手势。我们用了两种方法实现了，一个是使用指令，这个方法可以通过HTML模板看到，另一个是使用*$ionicGesture.on*方法实现的。
通常任何与DOM相关的代码都将写成一个（自定义的）指令。在这里，我们将他写在控制器里面以阐明目的。事件类型是*dragup*。我们使用*document.querySelector*API来取得DOM元素， 然后将此元素封装到一个AngularJS元素对象中。这个对象将作为*$ionicGesture.on*方法的第三个传入参数。
接下来，更改*www/index.html*的body部分：

```
<body ng-app="starter" ng-controller="AppCtrl">
    <ion-header-bar class="bar-dark">
        <h1 class="title">Gestures</h1>
    </ion-header-bar>
    <ion-content>
        <div class="card">
            <div id="gestureContainer" class="item text-center" on-drag-up="onDragUp()
">
            Drag me up!!
            </div>
        </div>
        <div class="card">
            <div class="item text-center">
            Scope Gesture : {{scopeGesture}}
            <br>
            Delegate Gesture : {{delegateGesture}}
            </div>
        </div>
    </ion-content>
</body>
```

我们创建了两个卡片容器；第一个包含了一个ID名为*gestureContainer*的*div*，他的*on-drag-up*属性将在事件触发的时候执行*onDragUp*方法。
第二个荣是是由*scopeGesture*的值组成的，这些值将会在*onDragUp*方法执行的时候进行更新。*delegateGesture*将在*$ionicGesture.on*方法发出*dragup*事件的时候进行设置。
保存所有文件，返回浏览器，可以看到这两个卡片部分。当你拖动第一个卡片内容的时候，第二个卡片的显示内容将会更新，如下：



管理手势非常简单！下面表格显示的是可用于*$ionicGesture.on*方法中的手势指令以及他的事件类型。可以用上述代码来实现以下任何一个手势：

Gesture name	Directive name	Event type (when using with `$ionicGesture.on()`)
Drag up	`on-drag-up`	`dragup`
Drag down	`on-drag-down`	`dragdown`
Drag right	`on-drag-right`	`dragright`
Drag left	`on-drag-left`	`dragleft`
Drag	`on-drag`	`drag`
Swipe up	`on-swipe-up`	`swipeup`
Swipe down	`on-swipe-down`	`swipedown`
Swipe right	`on-swipe-right`	`swiperight`
Swipe left	`on-swipe-left`	`swipeleft`
Swipe	`on-swipe`	`Swipe`
Hold (touch > 500ms)	`on-hold`	`hold`
Tap (touch < 250ms)	`on-tap`	`tap`
Double tap	`on-double-tap`	`doubletap`
Touch	`on-touch`	`touch`
Release	`on-release`	`release`

> 也可以使用*ionic.EventController*工具来实现手势操作。请参考：http://ionicframework.com/docs/api/utility/ionic.EventController/#onGesture 默认Ionic会移除浏览器添加的300ms点击延迟。浏览器添加这个延迟是用来区别单击与双击的。如果想要给某元素应用这个300ms的延迟，那么请使用*data-tap-disabled*属性。 更多信息，参考： http://ionicframework.com/docs/api/page/tap/

工具

本章最后的一个主题我们将用来探索Ionic提供的一些工具服务。首先要上的是*$ionicConfigProvider*。
Ionic默认是根据他的运行环境插入到应用配置里面的。并且，某些属性将根据环境选择性的去使用，例如transition。些这本书的时候， Ionic官方只支持Android和iOS。尽管如此，Ionic在其他平台上也是可以使用的。
所有配置都是基于app的运行的环境的。如果运行环境既不是Android也不是iOS，那么将默认给他使用iOS配置。
但是，我们还是可以通过*$ionicConfigProvider*服务来控制这些选择。可以通过如下方法来覆盖默认值：

```
.config(function ($ionicConfigProvider) {
    $ionicConfigProvider.views.transition('none');
    $ionicConfigProvider.views.maxCache(10);
    $ionicConfigProvider.form.checkbox('circle'); //square or circle
    $ionicConfigProvider.tabs.style('striped'); // striped or standard
    $ionicConfigProvider.templates.maxPrefetch(10);
    $ionicConfigProvider.navBar.alignTitle('right');
})
```

也可以通过以下方法为指定平台重写配置的默认值：

```
.config(function($ionicConfigProvider) {
    // Checkbox style. Android defaults to square and iOS defaults to circle.
    $ionicConfigProvider.platform.ios.form.checkbox('square');
    $ionicConfigProvider.platform.android.form.checkbox('circle');
})
```

> 可重写的属性请参考：
> http://ionicframework.com/docs/api/provider/$ionicConfigProvider/

Ionic通过*ionic.platform*提供了一系列的工具方法。可以使用这个对象提供的方法来检查环境信息：

```
.config(function() {
    console.log('ionic.Platform.isWebView()', ionic.Platform.isWebView());
    console.log('ionic.Platform.isIPad()',ionic.Platform.isIPad());
    console.log('ionic.Platform.isIOS()', ionic.Platform.isIOS());
    console.log('ionic.Platform.isAndroid()', ionic.Platform.isAndroid());
    console.log('ionic.Platform.isWindowsPhone()', ionic.Platform.isWindowsPhone());
})
```

> 其他*ionic.platform*方法请查看：http://ionicframework.com/docs/api/utility/ionic.Platform/

还有一些其他的方法帮你与DOM进行交互。这些方法在*ionic.DomUtil*对象里面可以找到。以下列举其中一些：

```
.controller('AppCtrl', function($scope) {
    var $element = angular.element(document.querySelector('#someElement'));
    console.log(ionic.DomUtil.getParentWithClass($element,'.card'));
    console.log(ionic.DomUtil.getParentOrSelfWithClass($element, '.card'));
    // requestAnimationFrame example
    function loop() {
        console.log('Animation Frame Requested');
        ionic.DomUtil.requestAnimationFrame(loop);
    }
    loop();
})
```

> 其他*ionic.DomUtil*方法请查看：http://ionicframework.com/docs/api/utility/ionic.DomUtil/

最后，我们来看一下Ionic的事件控制器（Event Controller）。他是由事件和手势的监听和移除监听的方法所组成的。你也可以使用*ionic.EventController*的trigger方法来触发事件。
以下部分展示了如何使用*ionic.EventController*来对事件和手势进行绑定，触发以及接触绑定的。
再次声明，以下逻辑需要去指令里面实现，然后在想要的元素上应用此指令：

```
.controller('AppCtrl', ['$scope', function($scope) {
    // 绑定事件
    var $body = document.querySelector('body');
    var eventListener = function() {
        console.log('Body Tapped!');
        ionic.EventController.off('tap', eventListener, $body);
    };
    ionic.EventController.on('tap', eventListener, $body);
    ionic.EventController.trigger('tap', {
        target: $body
    });

    // 绑定手势
    var cancelSwipeUp;
    var gestureListener = function() {
        console.log('Body Swiped Up!');
        ionic.EventController.offGesture(cancelSwipeUp, 'swipeup', gestureListener);
    }
    cancelSwipeUp = ionic.EventController.onGesture('swipeup', gestureListener, $body)
;
    ionic.EventController.trigger('swipeup', {
     target: $body
    });
}])
```

总结

本章中，我们学习了大量的Ionic指令和服务以帮助我们轻松的创建应用。我们先从Ionic Platform服务入手，然后进入到页头和页脚指令。 接下来，我们贯穿了所有内容相关（content-related）的指令和导航相关（navigation-related）的指令和服务。接着，我们学习了一下覆盖层（overlay）。 然后，我们快速的了解了列表指令，手势，以及工具服务。
完成这些之后，我们完成了对整个Ionic的熟悉过程。从下一章开始，我们将利用这些组件来制作简单和复杂的应用。
在下一章里，我们将制作一个书店（Book Store）应用，用户可以在其中进行注册和登入操作。用户可以通过浏览书的目录将他们添加到购物车。 用户也可以在他们的档案里面检出书籍以及查看购物历史。 这个应用展示了如何整合Ionic与一个安全的REST风格的后端服务。

第六章 制作一个书店应用

到目前为止，我们学习了所有Ionic的关键元素。本章中，我们将利用之前所学来制作一个书店应用。 本章的主要目的是巩固之前所学，同时，理解如何整合Ionic app与已知的REST服务。

> 记住，我们不会写任何服务端代码

我们将制作一个简单的多页面Ionic客户端，用来供用户浏览书籍，无需认证。只有在用户添加书籍到购物车，或者浏览订购历史记录的时候，我们才要求用户进行登录。 不强制用户要进行登录之后才浏览内容，而是在有需要的时候才进行登录认证，这样做可以带来更好的用户体验。
用户登录之后，就可以添加书籍到购物车，查看购物车，以及查看订单。应用的数据将会由一个使用JSON Web Token的安全的REST服务器管理。
开发过程中，我们将会学习以下主题：

- 学习应用的点对点架构
- 在本机设置服务器或者订购主机服务
- 分析应用将要用到的视图，控制器和工厂组件:6.3.1 6.3.2
- 应用视觉测试

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通:
> https://github.com/learning-ionic/Chapter-6

> 用到的名词：
>
> - end to end 端对端
> - end point 终端接口
> - request 请求
> - pagination 分页
> - data persistence layer 数据持久层
> - grid system 格子系统，栅格系统

书店应用的介绍

本章将开发一个书店应用。之前也说过，用户在其中可以进行注册和登录。用户可以在未经认证的情况下浏览书籍。 用户可以添加书籍到购物车，查看购物车，检出购物车。用户在有了购物记录之后，就可以浏览他们的订购历史页。
应用功能非常简单，但是想要提升应用的等级的话，就必须要给应用整合一个安全的REST API服务端，服务端使用的是Node.js开发，使用了JSON Web Token进行认证。 由于我们本机以及设置好了Node.js，所以设置服务器不会很复杂。如果你不想在本机搭建服务端的话，你也可以在下一章中找到一个外部主机的连接。
一些终端接口是使用**JSON Web Token**（**JWT)**加密的，例如检出和查看订购清单。服务器的设置方式是这样的：当一个TEST客户端（例如Ionic app）想要获取应用数据；那么它在请求的时候就必须发送有效的token。
应用的工作流是这样的：

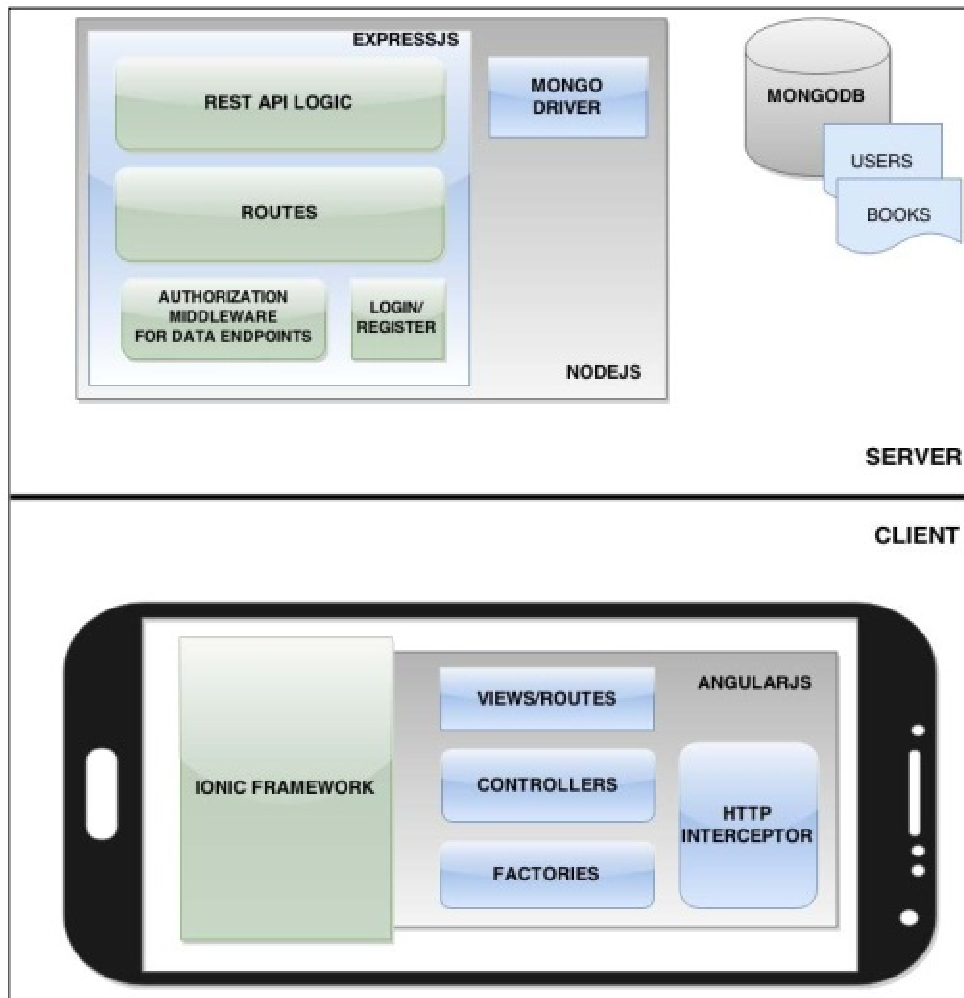
1. 用户启动app
2. 用户无需认证就可以浏览数据
3. 用户想要添加书籍到购物车或者查看订单
4. 用户想要注册或者登录
5. 注册或者登录成功，服务端将会返回一个有效期7天的token
6. 当客户端进行一些操作的时候（例如，添加到购物车，或者查看订单），它就会需要向对应的REST终端发送请求，请求中会包含token。 如果请求有效，并且对应当前用户，我们就给他分发数据；否则，我们将禁止用户访问或者更新数据。

下一个部分，我们将理解完整的应用架构。
我添加了一些额外的功能来更好的管理数据，例如分页，本地存储。但是没有100%的完成。这些代码片是给你示范如何整合功能到你的ionic app。

> 提醒：这不是一个产品级的app，但是可以作为制作产品级的开端。

书店应用的架构

以下是所有参与创建书店应用的组件的高级视图。



服务端架构

本应用使用的加密REST服务用的上Node.js/Express。数据持久层使用的是MongoDB。

> 应用中使用的数据是用Faker脚本生成的，Faker脚本：https://www.npmjs.com/package/faker 。本应用中的数据都是假的，仅可以用作应用原型，填满空白处。所有图片和文本都是随机的。

鉴于本书的目的是介绍Ionic，所以我不会讲解如何搭建服务端。

> 可以参考博客介绍服务端的帖子，了解如何设置，JWT是如何工作的。 参考搭建一个加密REST风格的Node.js应用

我搭建了两个相同实现的应用，一个是我使用Node.js作为服务端的Bucket列表应用，另一个是我使用Firebase作为服务端的同样的应用。 更多信息可以参考：

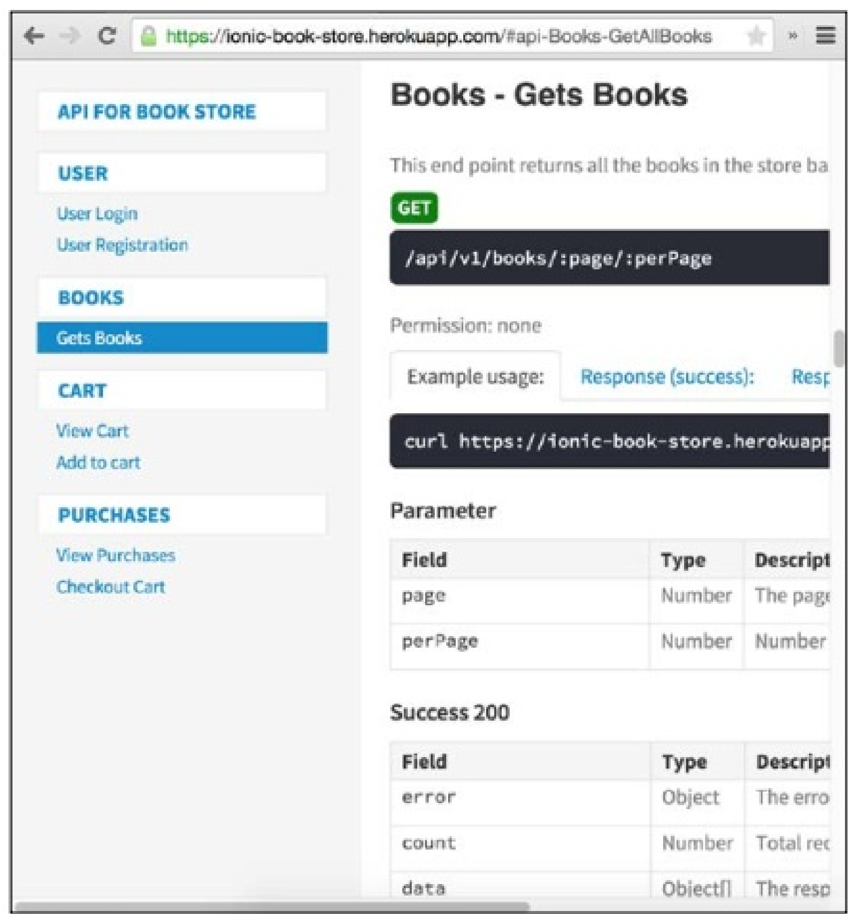
1. Ionic Restify MongoDB：端对端混合应用：http://thejackalofjavascript.com/an-end-to-end-hybrid-app/
2. 创建一个搭载Firebase的端对端Ionic 应用： http://www.sitepoint.com/creating-firebase-

powered-end-endionic-application/

服务端API文档

我提供了Bookstore REST API的文档，其中你可以查看所有的终端API，理解每个路由所需的输入数据，将会返回何种数据，可能会有哪些异常会发生。

> 文档在此处查看：https://ionic-bookstore.herokuapp.com/



文档的扩展性不强，但是详尽，简单易用。

客户端架构

客户端将会有以下路由：

- 主页，Home（查看所有书籍）
- 登录 Login（标签组件）/注册 Register（标签组件）
- 查看某本书
- 添加购物车
- 查看购物车
- 查看订单 我们将需要以下控制器：
- AppCtrl：应用级别的控制器（管理认证）

- BrowseCtrl：用来展示所有的书籍
- BookCtrl：用来显示某本书的详细信息
- CartCtrl：用来显示购物车
- PuerchasesCtrl：用来显示订单 我们将要用到4组工厂：一组管理Ionic加载，一组管理本地存储（localStorage），一组管理认证，一组管理数据。
- Loader：管理Ionic加载
- LSFactory：管理本地存储
- AuthFactory：管理认证
- TokenFacroty：管理每个HTTP请求的token
- BooksFactory：用来获取所有书籍
- UserFactory：用户的登录，注册以及购物车，订单的API

Github上的代码

服务端和客户端的代码都上传到Github了。建议检出代码至本机。我会添加更新以及修复读者反馈的任何bug。
同时鼓励读者在发现任何问题的时候发起issue，我将尽力完成：

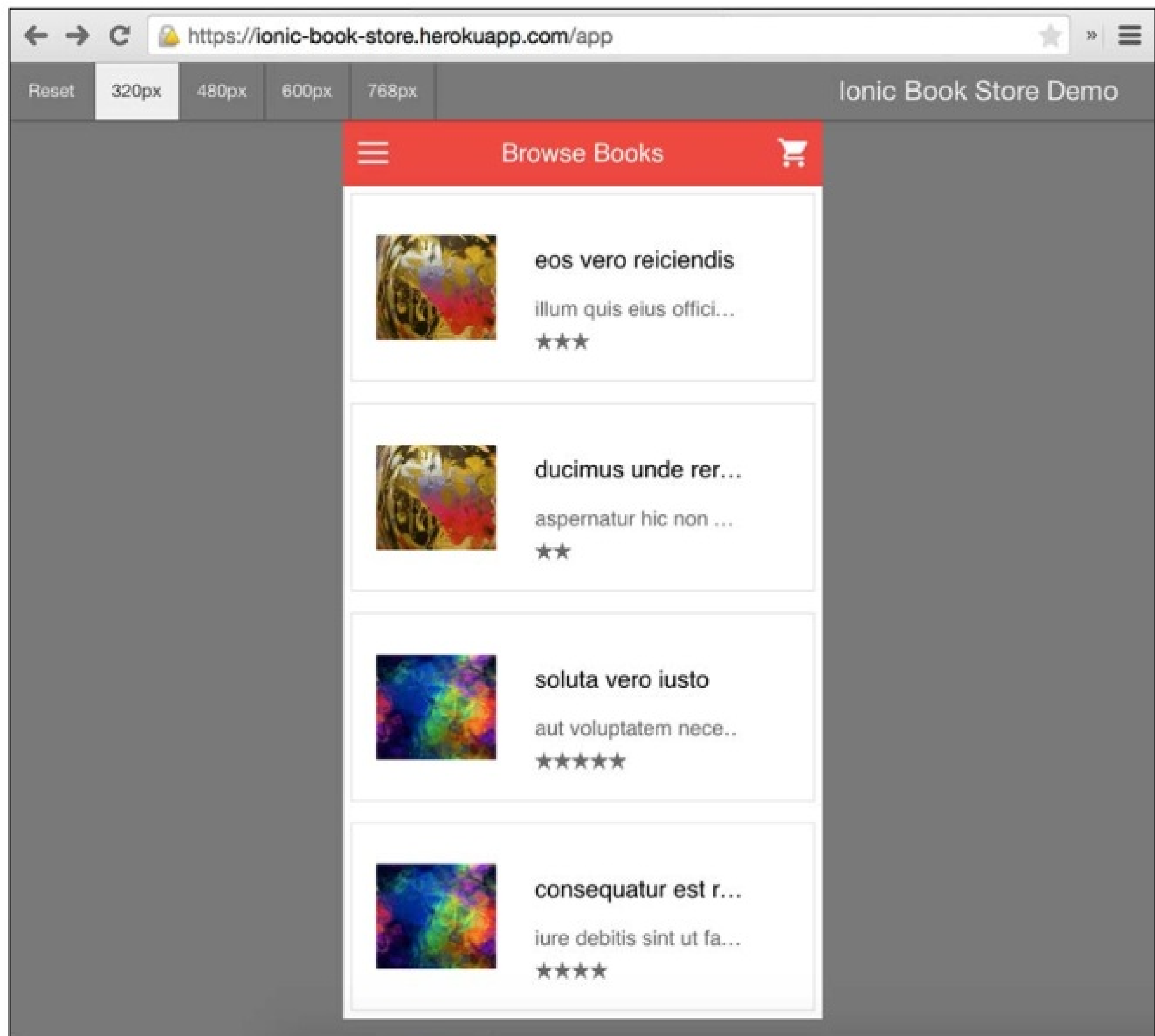
- Bookstore Ionic Client Repository
- Bookstore Node.js Server Repository

书店demo

这个书店app将使用侧边菜单模板来搭建。搭建本应用我们还会用到标签页，modal，加载，卡片，列表以及栅格系统。

> 在开始之前，可以提前预览一下我们将要搭建的app的效果： https://ionic-book-store.herokuapp.com/app

应用加载需要些时间，但是加载完成之后，就可以看到以下效果：



这是我们将要制作的应用的一个实时演示demo。可以随便点点看看其他页面的效果，例如菜单，例如购物车。 当你在添加购物车，或者访问购物车或者访问订单的时候，将会要求你进行注册或者登录。 你可以创建一个帐号进行测试。
顶部有一个分辨率条供你调整查看应用中不同分辨率下面的效果。

> 注意，此处展示的数据都是随机的假数据，仅供原型开发之用。

开发流

本章中，我们不会一个接一个的去开发功能，在完成一个之后再开发下一个。我们会一次性完成整个app的开发，然后查看输出效果。 如果你在开发某功能的时候有不懂的，想要知道最终会是什么样子，那么我强烈建议你检出实时版本的代码，也就是这个版本：https://ionic-book-store.herokuapp.com/app

设置服务器

鉴于设置服务器不是本书以及本应用的重点部分，以下提供两个方法来完成他：

1. 使用已有的REST API，也就是俺建的
2. 制作服务端代码分支，设置DB，本机运行服务器

对于第一个选项，可以参考文档 查看可用的REST API。
对于方法2，我们将使用以下操作：
从 *https://github.com/arvindr21* 下载服务端源代码并解压。如果你熟悉Node.js的话，那么你就会发现这一个典型的Express应用，其中使用了JWT中间件。
数据库方面，你可以使用本机的MongoDB，也可以使用免费的MongoLab（https://mongolab.com/）。对于本应用你也可以用我的MongoLab。记住，别人也在使用这个URL。
决定了连接方式之后，打开*server/db/connection.js*，更新第2行的连接；例如：

```
var db = mongojs('ionicbookstoreapp', ['users', 'books']);
```

接下来，在*db*文件夹里面打开终端/命令行，运行如下命令来生成一些测试数据：

```
node dbscript.js
```

这个脚本会帮你生成30本书以用在应用中。

> 如果你使用的是MongoLab URL的话，你就不用运行如上命令去生成书籍了。因为MongoLab已有这些数据了。

最后，使用*cd*命令进入*server*然后运行如下命令：

```
node server.js
```

这个命令将在*3000*端口上运行服务，然后你就可以通过 *http://localhost:3000* 来访问应用了。

> 当你导航到 *http://localhost:3000* 的发生错误的时候，不要惊慌。这是一个API服务器；因此主页上没有任何UI的。

打开 *http://localhost:3000/api/v1/books/1/10* 你会看到浏览器打印出来10本书的JSON数据。这意味这你的服务器搭建成功了。
再次声明，如果你对DIY（Do It Yourself 自做）的方式不适应的话，你可以直接使用已有的REST终端服务。我将给你展示使用已有服务主机是多么的简单，不用管REST API服务器在哪里。

搭建应用

现在，我们已经搭建好了服务器，并且对我们将要制作的东西有了一个初步的想法，那么我们现在就可以开始了。 执行以下步骤就可以轻松的搭建好这个app了：

1. 搭建侧边菜单模板
2. 重构此模板
3. 创建认证，本地存储以及REST API工厂
4. 为每个路由制作控制器，然后将它们整合到对应的工厂
5. 创建模板并整合控制器数据

第一步 - 搭建侧边菜单模板

第一件要做的事情就是搭建侧边菜单模板。新建一个文件夹名为*chapter6*。在*chapter6*文件夹内打开终端/命令行，运行如下命令：

```
ionic start -a "Ionic Book Store" -i app.bookstore.ionic book-store sidemenu
```

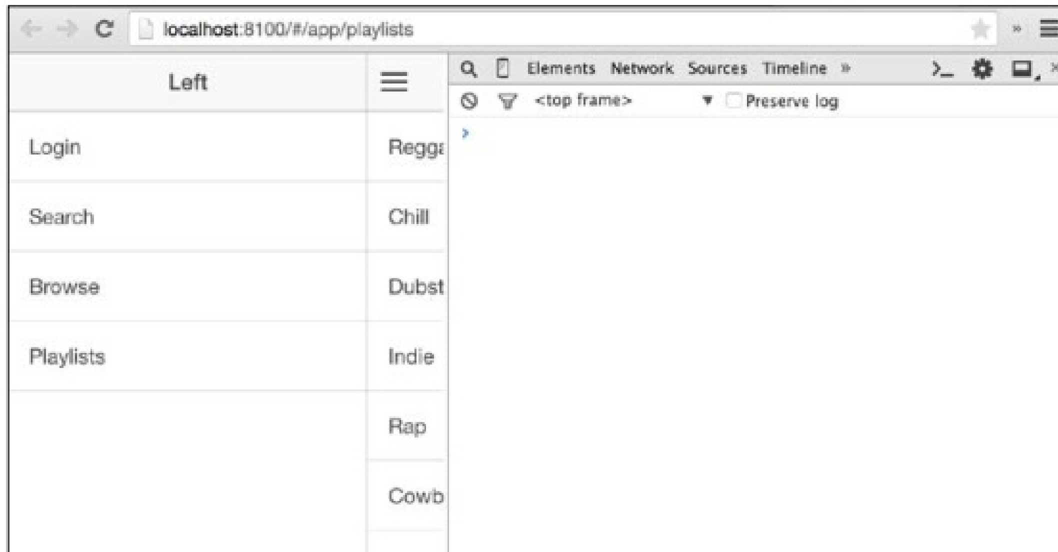
这条命令将会创建好一个侧边菜单模板应用。我们的应用不会自定义主题，所以不会设置SCSS。

第二步 - 重构模板

到目前为止的所有范例中，我们都只是使用部分组件。但是在本例中，我们将重构整个模板。
在继续进行之前，我们先确认一下一切是否正常。使用 *cd* 命令进入到 *book-store* 文件夹，然后运行：

```
ionic serve
```

这样，app就运行起来了。之前在 第二章 欢迎使用*Ionic* 提到过，打开开发工具，调整其位置到页面的右边部分。效果图如下：



重构菜单

第一个需要重构的是菜单。我们将使用需要用到的条目来替换默认的条目。
打开*www/templates/menu.html*，如下替换左边的*ion-side-menu*部分：

```
<ion-side-menu side="left">
    <ion-header-bar class="bar-assertive">
        <h1 class="title">Menu</h1>
    </ion-header-bar>
    <ion-content>
        <ion-list>
            <ion-item menu-close href="#/app/browse">
            Browse Books
            </ion-item>
            <ion-item menu-close href="#/app/cart">
            My Cart
            </ion-item>
            <ion-item menu-close href="#/app/purchases">
            My Purchases
            </ion-item>
            <ion-item menu-close ng-show="isAuthenticated" ng-click="logout()">
            Logout
            </ion-item>
            <ion-item menu-close ng-hide="isAuthenticated" ng-click="loginFromMenu()">
            Login
            </ion-item>
        </ion-list>
    </ion-content>
</ion-side-menu>
```

我们添加了4个菜单条目：

- Browse books：浏览所有书籍
- My cart：浏览购物车
- My purchases：浏览我的购物清单
- Login/Logout：根据用户授权状态条件展示的菜单

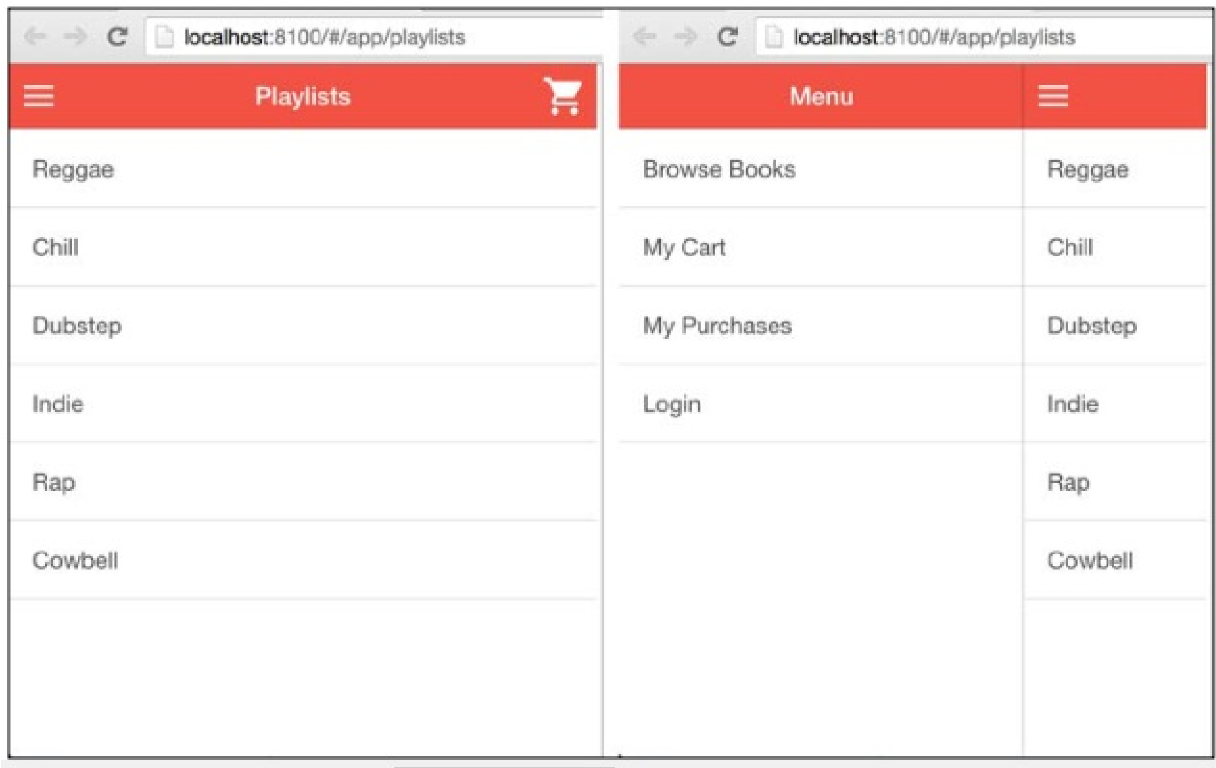
接下来，我们将更新*menu.html*的页头或者说*ion-side-menu-content*部分：

```
<ion-side-menu-content>
  <ion-nav-bar class="bar-assertive">
    <ion-nav-back-button>
    </ion-nav-back-button>
    <ion-nav-buttons side="left">
      <button class="button button-icon button-clear
            ion-navicon" menu-toggle="left">
        </button>
    </ion-nav-buttons>
    <ion-nav-buttons side="right">
      <!-- <button ng-show="isAuthenticated" class="button button-icon button-clear io
n-unlocked" ng-click="logout()">
    </button> -->
      <a class="button button-icon button-clear ionandroid-cart" href="#/app/cart">
      </a>
    </ion-nav-buttons>
  </ion-nav-bar>
  <ion-nav-view name="menuContent"></ion-nav-view>
</ion-side-menu-content>
```

我们将页头的主题修改为成assertive心情。同时，我们也在右边添加了两个按钮。一个是登出，这个按钮注释掉了，另一个是购物车。 这些代码教会你如何给应用的页头添加图标。

> 同时也需要注意到登出图标是一个条件显示图标。只有当用户在授权的状态下才显示。由于我不想再页头右边出现两个按钮，所以我把它注释掉了。

同时将*ion-side-menu*上的*enable-menu-with-back-views*属性设置为*true*。没有设置的话，你在子视图里面将看不到侧边菜单图标。
当你保存文件，返回浏览器的时候，将会看到如下结果：



重构模组名

接下来，我们将要给模组重命名。默认开始模板的模组名叫做*starter*。我们会将他重命名为*BookStoreApp*。需要更改的位置有：

- *www/index.html*里面的*ng-app*指令的名字改为*BookStoreApp*
- 打开*www/js/app.js*，将angular模组改成这样： **angular.module('BookStoreApp', ['ionic', 'BookStoreApp.controllers'])**
- 打开*www/js/controller.js*，更新*starter.controllers*为*BookStoreApp.controllers* 这样就完成了模组的重命名。从现在开始，在app中我们就可以是使用*BookStoreApp*作为模组的命名空间了。

添加run方法，修改路由

接着，我们将进行路由的配置；打开 *www/js/app.js* 。删除当中的*config*部分。接下来，我们将设置*run*方法来持有一些工具方法。 在*www/js/app.js*添加以下代码：

```
.run(['$rootScope', 'AuthFactory',function($rootScope, , AuthFactory) {
    $rootScope.isAuthenticated = AuthFactory.isLoggedIn();

    // utility method to convert number to an array of elements
    $rootScope.getNumber = function(num) {
        return new Array(num);
    }
}
])
```

> 可以给一个模组添加多个 run 方法

如果你还记得我们在*menu.html*添加的根据*isAuthenticated*条件显示的登录和登出按钮的话，*isAuthenticated*属性就是在此处初始化的。 我们也有一个工具方法名为*getNumber*。这个方法是用来以传入的参数作为长度生成一个数组的。
接下来，我们将添加路由。在*run*之下，添加如下*config*代码：

```
.config(['$stateProvider', '$urlRouterProvider', '$httpProvider', function($stateProvi
der, $urlRouterProvider, $httpProvider) {
    // setup the token interceptor
    $httpProvider.interceptors.push('TokenInterceptor');

    $stateProvider
        .state('app', {
            url: "/app",
            abstract: true,
            templateUrl: "templates/menu.html",
            controller: 'AppCtrl'
        })
        .state('app.browse', {
            url: "/browse",
            views: {
                'menuContent': {
                    templateUrl: "templates/browse.html",
                    controller: 'BrowseCtrl'
                }
            }
        })
        .state('app.book', {
            url: "/book/:bookId",
            views: {
                'menuContent': {
                    templateUrl: "templates/book.html",
                    controller: 'BookCtrl'
                }
            }
        })
        .state('app.cart', {
            url: "/cart",
            views: {
```

```
                'menuContent': {
                    templateUrl: "templates/cart.html",
                    controller: 'CartCtrl'
                }
                }
        })
        .state('app.purchases', {
            url: "/purchases",
            views: {
                'menuContent': {
                    templateUrl: "templates/purchases.html",
                    controller: 'PurchasesCtrl'
                }
            }
        });
    // if none of the above states are matched, use this as    the fallback
    $urlRouterProvider.otherwise('/app/browse');
}
])
```

> 虽然我们可以手动编辑路由，但是我认为这样展示更简单。 同时，如果你的Ionic服务一直在运行的时候，你应该会在浏览器的JavaScript控制台看到错误输出：找不到依赖库。在所有代码完成之前，app都不会正常工作。可以在所有代码完成之前关闭服务器。

config里面我们添加的第一个是*TokenInterceptor*。当我们在使用认证工厂的时候，我们会实现*TokenInterceptor*。在使用*TokenInterceptor*的时候，我会寻回此处。
我们添加了以下这些路由：

- /app：这是用来管理应用主视图的一个抽象路由
- /browse：这是应用的主页，用来列出所有书籍
- /book/:bookId：展示某本书的详情
- /purchase：展示用户过往的订购记录。 登录和注册标签界面将会是一个modal而不是一个route。

重构模板

我们可以根据路由配置一次重构一个模板，或者为了便利起见，我们可以（或者说将要）删除 *www/templates* 里面除了 *menu.html*之外的所有模板，然后添加一些空白的HTML文件。
删除除了*menu.html*之外的所有模板之后，我们创建以下几个模板：

- browse.html
- book.html
- cart.html
- purchases.html
- login.html 目前可以将以上这些模板都留空。最后我们才会用到这些模板。

第三步 - 创建认证，本地存储和REST API工厂

在*www/js*文件夹内，创建一个文件名为*factory.js*。然后在*www/index.html*的*controller.js*的引用后面添加：

```
<script src="js/factory.js"></script>
```

这个文件将用来持有所有的工厂。当然，如果你喜欢的话你也可以给每个功能添加一个工厂文件。但是咱们的范例里面所有工厂都在一个文件里。
文件第一行要添加的是REST API的基本URL。这个URL可以指向你的本地服务器也可以指向Heroku上的已有REST API地址。如下：

```
//var base = 'http://localhost:3000';
var base = 'https://ionic-book-store.herokuapp.com';
```

可以根据需求来注释其中一行。
接下来创建一个新的模组名为*BookStoreApp.factory*来持有所有的工厂定义。
angular.module('BookStoreApp.factory', [])
然后在主模组 *BookStoreApp* 中将此模组添加为依赖。打开 *www/js/app.js* ，更新*BookStoreApp*如下：
angular.module('BookStoreApp', ['ionic', 'BookStoreApp.controllers', 'BookStoreApp.factory'])

Ionic loading 工厂

第一个制作的工厂是加载工厂。在AngularJS模组定义下，添加如下：

```
.factory('Loader', ['$ionicLoading', '$timeout', function($ionicLoading, $timeout) {
    var LOADERAPI = {
        showLoading: function(text) {
            text = text || 'Loading...';
            $ionicLoading.show({
                template: text
            });
        },

        hideLoading: function() {
            $ionicLoading.hide();
        },

        toggleLoadingWithMessage: function(text, timeout) {
            $rootScope.showLoading(text);

            $timeout(function() {
                $rootScope.hideLoading();
            }, timeout || 3000);
        }
    };
    return LOADERAPI;
}])
```

我们有以下三个方法来帮助我们展示，隐藏和切换加载：

- showLoading：显示一个带文本或者*Loading*的阻断modal
- hideLoading：隐藏使用*showLoading*显示的modal
- toggleLoadingWithMessage：使用提供的信息显示modal，并且在提供的超时或者默认的3秒后隐藏。这个方法将用来展示自动隐藏的阻断信息。

localStorage工厂

接下来我们要添加*localStorage*工厂。在Loader工厂之前，添加如下：

```
.factory('LSFactory', [function() {
    var LSAPI = {
        clear: function() {
            return localStorage.clear();
        },

        get: function(key) {
            return JSON.parse(localStorage.getItem(key));
        },

        set: function(key, data) {
            return localStorage.setItem(key, JSON.stringify(data));
        },

        delete: function(key) {
            return localStorage.removeItem(key);
        },

        getAll: function() {
            var books = [];
            var items = Object.keys(localStorage);
            for (var i = 0; i < items.length; i++) {
                if (items[i] !== 'user' || items[i] != 'token') {
                    books.push(JSON.parse(localStorage[items[i]]));
                }
            }
            return books;
        }
    };
    return LSAPI;
}])
```

这个工厂提供了我们和HTML5 *localStorage* 交互的API。我们将使用本地存储来保存所有书籍。这样一来，我们不需要每次获取书籍的时候都调用服务端接口。
由于这是个范例而已，我们就不会在后台检查新书籍然后更新本地缓存。
loacalStorage 的 *clear*, *getItem*,*setItem*he *deleteItem*方法分别封装成了*clear*，*get*，*set*和*delete*。由于我们需要将对象存放到本地存储，而本地存储之支持字符串， 所以我们在存数据之前将对象字符串话，在取出来之后对象化。
*getAll*方法将用来获取我们在*localStorage*中存放的所有书籍。

> 注意我们使用了*Object.keys*来将*localStorage*（类数组）转换成一个数组。 更多信息关于*Object.keys*请参考：https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys

认证工厂

下一个进行的是Authentication（认证）工厂。在*LSFactory*定义后面添加以下代码：

```
.factory('AuthFactory', ['LSFactory', function(LSFactory) {
    var userKey = 'user';
    var tokenKey = 'token';
    var AuthAPI = {
        isLoggedIn: function() {
            return this.getUser() === null ? false : true;
        },

        getUser: function() {
            return LSFactory.get(userKey);
        },

        setUser: function(user) {
            return LSFactory.set(userKey, user);
        },

        getToken: function() {
            return LSFactory.get(tokenKey);
        },

        setToken: function(token) {
            return LSFactory.set(tokenKey, token);
        },
        deleteAuth: function() {
            LSFactory.delete(userKey);
            LSFactory.delete(tokenKey);
        }
    };
    return AuthAPI;
}])
```

此工厂依赖*LSFactory*，用来管理用户认证数据。就像之前说的那样，当用户注册或者登录的时候，服务端在返回用户数据的同时附带了访问token。 这些封装好了的方法通过*LSFactory*的API保存用户数据和token到本地存储。

接下来，我们将创建*TokenInterceptor*工厂。如果你还记得的话，我们之前在*www/js/app.js*的config部分中已经添加了*TokenInterceptor*的引用。 我们告诉AngularJS在他每次发起HTTP请求的时候调用*TokenInterceptor*。

> 什么是拦截器（**Interceptors**）？ 当使用***$http***服务发起**AJAX**请求的时候，我们的**app**里面有些时候需要在发起**HTTP**请求之前添加一些额外的数据。 **AngularJS**中这种用途的代码就叫做拦截器。 将拦截器添加到***$httpProvider***的语法是这样的：$httpProvider.interceptors.push('MyInterceptor'); * 在设置好名为MyInterceptor* 的工厂或者服务之后，可以参考此处获取更多拦截器和他的工作流程的信息：
> http://www.webdeveasy.com/interceptors-in-angularjs-anduseful-examples/

当调用此方法的时候，他会检查*localStorage*中是否有用户数据和用户token。如果有的话他会将这些数据添加到请求头里面。如果你还记得的话，添加购物车，查看购物车，检出，以及查看订单路由都需要用户认证。 token是服务端知道用户是否认证成功是有授权浏览相关内容的唯一途径。

在*www/js/factory.js*的*AuthFactory*的下面，添加*TokenInterceptor*工厂的代码如下：

```
.factory('TokenInterceptor', ['$q', 'AuthFactory', function($q, AuthFactory) {
    return {
        request: function(config) {
            config.headers = config.headers || {};
            var token = AuthFactory.getToken();
            var user = AuthFactory.getUser();

            if (token && user) {
                config.headers['X-Access-Token'] = token.token;
                config.headers['X-Key'] = user.email;
                config.headers['Content-Type'] =
                "application/json";
            }
            return config || $q.when(config);
        },
        response: function(response) {
            return response || $q.when(response);
        }
    };
}])
```

REST API工厂

接下来，我们要创建两个工厂。一个用作与REST API服务终端交互，无需认证就可以拿到数据，另一个工厂用作与REST API服务终端交互，需要认证。这只是一个逻辑分离。可以在一个工厂里面操作。

第一个是*BookFactory*。其中有一个*get*方法与公共的*api/v1/books*终端对话：

```
.factory('BooksFactory', ['$http', function($http) {
    var perPage = 30;
    var API = {
        get: function(page) {
            return $http.get(base + '/api/v1/books/' + page + '/' + perPage);
        }
    };
    return API;
}])
```

api/v1/books REST 终端服务有数据分页的能力。你可以发送*perPage*和*page*数字；这样他将会发挥对应的记录。 你可以使用这个API来请求分页数据，然后更具需求加载书籍。
但是，在本应用中，我们只请求1次，返回30条记录。你可以实现一个*ion-infinite-scroll*根据需求加载书籍作为练习。
接下来是*UserFactory*。这个工厂是由*login*，*register* REST终端以及其他4个购物车和订购相关的API组成。
在*BooksFactory*的后面添加*UserFactory*的相关定义：

```
.factory('UserFactory', ['$http', 'AuthFactory',
    function($http, AuthFactory) {
        var UserAPI = {
            login: function(user) {
                return $http.post(base + '/login', user);
            },

            register: function(user) {
                return $http.post(base + '/register', user);
            },

            logout: function() {
                AuthFactory.deleteAuth();
            },

            getCartItems: function() {
                var userId = AuthFactory.getUser()._id;
                return $http.get(base + '/api/v1/users/' + userId + '/cart');
            },

            addToCart: function(book) {
                var userId = AuthFactory.getUser()._id;
                return $http.post(base + '/api/v1/users/' + userId + '/cart', book);
            },

            getPurchases: function() {
                var userId = AuthFactory.getUser()._id;
                return $http.get(base + '/api/v1/users/' + userId + '/purchases');
            },

            addPurchase: function(cart) {
                var userId = AuthFactory.getUser()._id;
                return $http.post(base + '/api/v1/users/' + userId + '/purchases', car
t);
            }
        };
        return UserAPI;
    }
])
```

getCartItems，*addToCart*，*getPurchases*和*addPurchase*方法在REST调用之前会先从*localStorage*里面获取用户ID。REST终端服务需要的也是这样的URL。

完成这些之后，我们成功的加入了工厂方法来管理数据，认证以及REST交互。在处理相关控制器的时候，我们会讲解每个REST终端的响应。

如果你的服务器还在运行并且你正确的做完所有事情的话，你应该会看到一个关于*BrowserCtrl*的错误。这样就是对的，我们接下来就处理控制器了。

第四步 - 给每个路由创建控制器并将其整合到工厂

现在，我们工厂配置好，我们就要创建所需的控制器了。再一次，为了简单起见，我们删掉了*www/js/controller.js*里面的*AppCtrl*，*PlaylistsCtrl*，和*PlayListCtrl*，只留下：**angular.module('BookStoreApp.controllers', [])** 。

应用控制器

第一个要添加的控制器是应用控制器或者*AppCtrl*。这个控制器用来管理登录和注册功能。如果其他子控制器（例如购物车控制器）想要强制用户登录， 在处理请求之前，这个控制器将会在scope内广播一个*showLoginModal*，然后*showLoginModal*的监听器将会负责管理登录和注册流程。一旦用户登录成功，将会通知购物车控制器继续操作。
我们将利用*$on* 和 *$broadcast*来触发自定义事件，例如：*showLoginModal* 。
我们将在*www/js/controller.js*的模组定义后面添加*AppCtrl*。为更好的解释此代码，我们将*AppCtrl*代码分离成两部分。
首先，添加定义与所有需要的依赖：

```
.controller('AppCtrl', ['$rootScope', '$ionicModal','AuthFactory', '$location', 'UserF
actory', '$scope', 'Loader',
function($rootScope, $ionicModal, AuthFactory, $location,UserFactory, $scope, Loader)
{
}])
```

接下来，在root scope（根范围）内注册*showLoginModal*事件。如下：

```
$rootScope.$on('showLoginModal', function($event, scope, cancelCallback, callback) {
    $scope.user = {
        email: '',
        password: ''
    };
    $scope = scope || $scope;
    $scope.viewLogin = true;
    $ionicModal.fromTemplateUrl('templates/login.html', {
        scope: $scope
    }).then(function(modal) {
        $scope.modal = modal;
        $scope.modal.show();
        $scope.switchTab = function(tab) {
            if (tab === 'login') {
                $scope.viewLogin = true;
            } else {
                $scope.viewLogin = false;
            }
        }
```

```
        $scope.hide = function() {
            $scope.modal.hide();
            if (typeof cancelCallback === 'function') {
                cancelCallback();
            }
        }
        $scope.login = function() {
            Loader.showLoading('Authenticating...');
            UserFactory.login($scope.user).success(function(data) {
                data = data.data;
                AuthFactory.setUser(data.user);
                AuthFactory.setToken({
                    token: data.token,
                    expires: data.expires
                });
                $rootScope.isAuthenticated = true;
                $scope.modal.hide();
                Loader.hideLoading();

                if (typeof callback === 'function') {
                    callback();
                }

            }).error(function(err, statusCode) {
                Loader.hideLoading();
                Loader.toggleLoadingWithMessage(err.message);
            });
        }
        $scope.register = function() {
            Loader.showLoading('Registering...');
            UserFactory.register($scope.user).
            success(function(data) {
                data = data.data;
                AuthFactory.setUser(data.user);
                AuthFactory.setToken({
                    token: data.token,
                    expires: data.expires
                });
                $rootScope.isAuthenticated = true;
                Loader.hideLoading();
                $scope.modal.hide();
                if (typeof callback === 'function') {
                    callback();
                }
            }).error(function(err, statusCode) {
                Loader.hideLoading();
                Loader.toggleLoadingWithMessage(err.message);
            });
        }
    });
});
```

接下来，在*$rootScope*属性上加上两个方法。这两个方法将在*sidemenu*中调用：

```
$rootScope.loginFromMenu = function() {
    $rootScope.$broadcast('showLoginModal', $scope, null, null);
}

$rootScope.logout = function() {
    UserFactory.logout();
    $rootScope.isAuthenticated = false;
    $location.path('/app/browse');
    Loader.toggleLoadingWithMessage('Successfully Logged Out!', 2000);
}
```

*showLoginModal*接受3个参数：

- scope：modal会在哪个scope里面创建。如果没有提供scope的话，就会使用*AppCtrl*。
- cacelCallback：当用户取消登录或者注册流程的时候执行的回调函数。
- callback：注册或者登录成功之后的回调函数。

在 *login.html* 文件里面，我们使用 *$ionicModal* 服务创建了一个modal。我们将在下面的部分使用这个模板。
login 和 *register* 方法负责与 *UserFactory* 里面对应的 *login* 和 *register* 方法对话。当用户登录或者注册成功的时候，服务端返回的结果看起来应该是这样子的：

```
{
    "error": null,
    "data": {
        "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE0
    MzM1MjIwNzQ4MzYsInVzZXIiOnsiX2lkIjoiNTU2ODU5NjgyN2ZjY2JjMTZkNjA4MzBi
    IiwiZW1haWwiOiJhQGEuY29tIiwibmFtZSI6ImEiLCJjYXJ0IjpbeyJpZCI6IjU1NjgzY
    2Q4ZmJiMmUxOTI0ZjE4YTRlYSIsInF0eSI6MX1dLCJwdXJjaGFzZXMiOlt7IlB1cmNoYX
    NlIG1hZGUgb24gMjktTWF5LTIwMTUgYXQgMTc6NTAiOlt7ImlkIjoiNTU2ODNjZDhmYmI
    yZTE5MjRmMThhNGU4IiwicXR5IjoxfV19LHsiUHVyY2hhc2UgbWFkZSBvbiAyOS1NYXkt
    MjAxNSBhdCAxNzo1OSI6W3siaWQiOiI1NTY4M2NkOGZiYjJlMTkyNGYxOGE0ZWIiLCJxd
    HkiOjF9LHsiaWQiOiI1NTY4M2NkOGZiYjJlMTkyNGYxOGE0ZjYiLCJxdHkiOjF9LHsiaW
    QiOiI1NTY4M2NkOGZiYjJlMTkyNGYxOGE0ZjgiLCJxdHkiOjF9XX0seyJQdXJjaGFzZSB
    tYWRlIG9uIDI5LU1heS0yMDE1IGF0IDIwOjUxIjpbeyJpZCI6IjU1NjgzY2Q4ZmJiMmUx
    OTI0ZjE4YTRlOCIsInF0eSI6MX0seyJpZCI6IjU1NjgzY2Q4ZmJiMmUxOTI0ZjE4YTRlY
    SIsInF0eSI6MX0seyJpZCI6IjU1NjgzY2Q4ZmJiMmUxOTI0ZjE4YTRlZSIsInF0eSI6MX
    1dfV19fQ.J0U0BZXhP6C1VWEHDT18BMOzkK_dvXP-xdGUiIr7-z8",
        "expires": 1433522074836,
        "user": {
            "_id": "5568596827fccbc16d60830b",
            "email": "a@a.com",
            "name": "a",
        }
    },
    "message": "Success"
}
```

我们从响应数据中提取 *token* 和 *user* 对象，然后使用*AuthFactory*的*setUser*和*setToken*方法将它们设入*localStorage*。*TokenInterceptor*将在进行REST调用之前在其中读取数据附加到请求中。

loginFromMenu 和 *logout* 是由菜单条目*Login/Logout*发起调用的。*loginFromMenu*所做的只是广播*showLoginModal*来处理用户认证。

*logout*方法调用*UserFactory*的登录方法，此方法用来清理*localStorage*中的用户数据和token。 同时他也会重置*isAuthenticated*属性，并将用户重定向到*/app/browse*页面。

浏览控制器（**browse controller**）

接下来需要处理的是*BrowseCtrl*。这个控制器负责在用户启动应用的时候展示所有书籍。*BrowseCtrl*在首次和REST API终端服务成功拿取数据之后将它们存放到*localStorage*中。此后，每次调用之前，都会在*localStorage*查找，节约电池，节省流量。

> 作为练习，实现一个后台程序用最新的书籍来更新*localStorage*

```
.controller('BrowseCtrl', ['$scope', 'BooksFactory', 'LSFactory','Loader', function($s
cope, BooksFactory, LSFactory, Loader) {
    Loader.showLoading();
    // support for pagination
    var page = 1;
    $scope.books = [];
    var books = LSFactory.getAll();
    // if books exists in localStorage, use that instead of making a call
    if (books.length > 0) {
        $scope.books = books;
        Loader.hideLoading();
    } else {
        BooksFactory.get(page).success(function(data) {
            // process books and store them
            // in localStorage so we can work with them later on,
            // when the user is offline
            processBooks(data.data.books);
            $scope.books = data.data.books;
            $scope.$broadcast('scroll.infiniteScrollComplete');
            Loader.hideLoading();
        }).error(function(err, statusCode) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage(err.message);
        });
    }
    function processBooks(books) {
        LSFactory.clear();
        // we want to save each book individually
        // this way we can access each book info. by it's _id
        for (var i = 0; i < books.length; i++) {
            LSFactory.set(books[i]._id, books[i]);
        };
    }
}
])
```

鉴于咱们REST API支持分页，我们需要传入*page*和*BookFactory*里面设置好的*perPage*来获取分页数据。我将页数动态化，你可以根据需求和分页加载书籍。
但是本范例中我们将一次加载所有的30本书。

书籍控制器（**book controller**）

用户浏览了所有书籍然后想要查看某特定书籍的详情的时候，他将点击书籍。此时，我们将用户重定向到*/book*页，在这里将会触发*BookCtrl*。 *BookCtrl*将在*localStorage*中查找指定id的书籍然后展示他的详情。
在*www/js/controllers.js*的*BrowseCtrl*后面加上*BookCtrl*代码：

```
.controller('BookCtrl', ['$scope', '$state', 'LSFactory','AuthFactory', '$rootScope',
'UserFactory', 'Loader',
  function($scope, $state, LSFactory, AuthFactory, $rootScope, UserFactory, Loader) {
    var bookId = $state.params.bookId;
    $scope.book = LSFactory.get(bookId);
    $scope.$on('addToCart', function() {
        Loader.showLoading('Adding to Cart..');
        UserFactory.addToCart({
            id: bookId,
            qty: 1
        }).success(function(data) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage('Successfully
            added ' + $scope.book.title + ' to your cart', 2000);
        }).error(function(err, statusCode) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage(err.message);
        });
    });
    $scope.addToCart = function() {
        if (!AuthFactory.isLoggedIn()) {
            $rootScope.$broadcast('showLoginModal', $scope,  null, function() {
                // user is now logged in
                $scope.$broadcast('addToCart');
            });
            return;
        }
        $scope.$broadcast('addToCart');
    }
}
])
```

我们使用*LSFactory.get(bookId)*从*localStorage*中获取书籍。
查看demo的时候，你会发现在书籍详情页面上有一个**Add to Cart**（添加到购物车）的按钮。添加到购物车的逻辑非常简单。用户在点击**Add to Cart**的时候， 我们检查用户时候已经登录。如果没有，我们广播一个*showLoginModal*事件，尔后在用户登录成功之后广播*addToCart*事件，这样就可以将当前展示的书籍添加到购物车。
如果用户已经登录，那么直接广播*addToCart*。
*addToCart*事件创建了一个对象，含有*bookid*和数量属性（硬编码到1），然后调用*UserFactory.addToCart*方法。这个方法负责添加书籍到购物车。

购物车控制器（cart controller）

接下来就是购物车控制器了。这个控制器是在用户点击页头的购物车图标或者点击侧边菜单的购物车按钮的时候调用的。
当用户点击购物车的时候，我们会检查用户是否是登录状态。如果用户是登录状态，我们将获取用户购物车条目并展示给用户看。如果没有登录的话就广播*showLoginModal*，这个事件

负责处理认证。 如果用户取消登录modal，我们会将用户带回*/app/browse*。
当用户认证成功或者登录成功的时候，我们将广播*getCart*事件。此事件将调用*UserFactory.getCartItems*方法获取所有的购物车条目。获取成功之后，我们会将他们赋值给*$scope*的*books*属性。
在*BookCtrl*后面添加*CartCtrl*代码：

```
.controller('CartCtrl', ['$scope', 'AuthFactory', '$rootScope','$location', '$timeout'
, 'UserFactory', 'Loader',
  function($scope, AuthFactory, $rootScope, $location, $timeout,UserFactory, Loader) {
    $scope.$on('getCart', function() {
        Loader.showLoading('Fetching Your Cart..');
        UserFactory.getCartItems().success(function(data) {
            $scope.books = data.data;
            Loader.hideLoading();
        }).error(function(err, statusCode) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage(err.message);
        });
    });
    if (!AuthFactory.isLoggedIn()) {
        $rootScope.$broadcast('showLoginModal', $scope, function()
        {
            // cancel auth callback
            $timeout(function() {
                $location.path('/app/browse');
            }, 200);
        }, function() {
            // user is now logged in
            $scope.$broadcast('getCart');
        });
        return;
    }
    $scope.$broadcast('getCart');
    $scope.checkout = function() {
        // we need to send only the id and qty
        var _cart = $scope.books;
        var cart = [];
        for (var i = 0; i < _cart.length; i++) {
            cart.push({
                id: _cart[i]._id,
                qty: 1 // hardcoded to 1
            });
        };
        Loader.showLoading('Checking out..');
        UserFactory.addPurchase(cart).success(function(data) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage('Successfully checked  out', 2000);
            $scope.books = [];
        }).error(function(err, statusCode) {
            Loader.hideLoading();
            Loader.toggleLoadingWithMessage(err.message);
        });
    }
}
])
```

我们在*$scope*上添加了*checkout*方法。当用户查看他的购物车的时候，他们可以检出购物车，此方法会调用*UserFactory.addPurchase*方法，并传入*cart*数组。 *cart*数组是由添加到购物的书籍的*id*和*quantity*组成的对象所组成的。

订购控制器（purchase controller）

咱们应用的最后一个控制器是订购控制器。这个控制器展示了当前登入用户的订购清单。跟购物车控制器一样，我们要先检查用户是否已经登录之后才能继续。 一旦用户登录成功之后，我们广播*getPurchases*事件。这个事件将负责与*UserFactory.getPurchases*方法联络，获取订购清单：

```
.controller('PurchasesCtrl', ['$scope', '$rootScope','AuthFactory', 'UserFactory', '$t
imeout', 'Loader',
  function($scope, $rootScope, AuthFactory, UserFactory,$timeout, Loader) {
    // http://forum.ionicframework.com/t/expandable-list-inionic/3297/2
    $scope.groups = [];

    $scope.toggleGroup = function(group) {
        if ($scope.isGroupShown(group)) {
            $scope.shownGroup = null;
        } else {
            $scope.shownGroup = group;
        }
    };

    $scope.isGroupShown = function(group) {
        return $scope.shownGroup === group;
    };

    $scope.$on('getPurchases', function() {
        Loader.showLoading('Fetching Your Purchases');
        UserFactory.getPurchases().success(function(data) {
            var purchases = data.data;
            $scope.purchases = [];
            for (var i = 0; i < purchases.length; i++) {
                var key = Object.keys(purchases[i]);
                $scope.purchases.push(key[0]);
                $scope.groups[i] = {
                    name: key[0],
                    items: purchases[i][key]
                }
                var sum = 0;
                for (var j = 0; j < purchases[i][key].length; j++) {
                    sum +=
                    parseInt(purchases[i][key][j].price);
                };
                $scope.groups[i].total = sum;
            };
```

```
                Loader.hideLoading();
            }).error(function(err, statusCode) {
                Loader.hideLoading();
                Loader.toggleLoadingWithMessage(err.message);
            });
        });

        if (!AuthFactory.isLoggedIn()) {
            $rootScope.$broadcast('showLoginModal', $scope,    function() {
                $timeout(function() {
                    $location.path('/app/browse');
                }, 200);
            }, function() {
                // user is now logged in
                $scope.$broadcast('getPurchases');
            });
            return;
        }
        $scope.$broadcast('getPurchases');
    }
    ])
```

我们添加了两个方法，*toggleGroup*和*isGroupShown*。这两个方法将在我们建立模块的时候调用。一旦响应返回的时候，我们将格式化这些数据，这样他就可以被组成一个内嵌列表。

> 建议在继续学习之前检出此app的在线版本以查看订购功能。这样你就可以理解此部分代码做了些什么。

第五步 - 创建模板并使用控制器数据进行整合

现在我们的控制器准备好了数据，那么我们将创建模板来展示这些数据。

登录模板

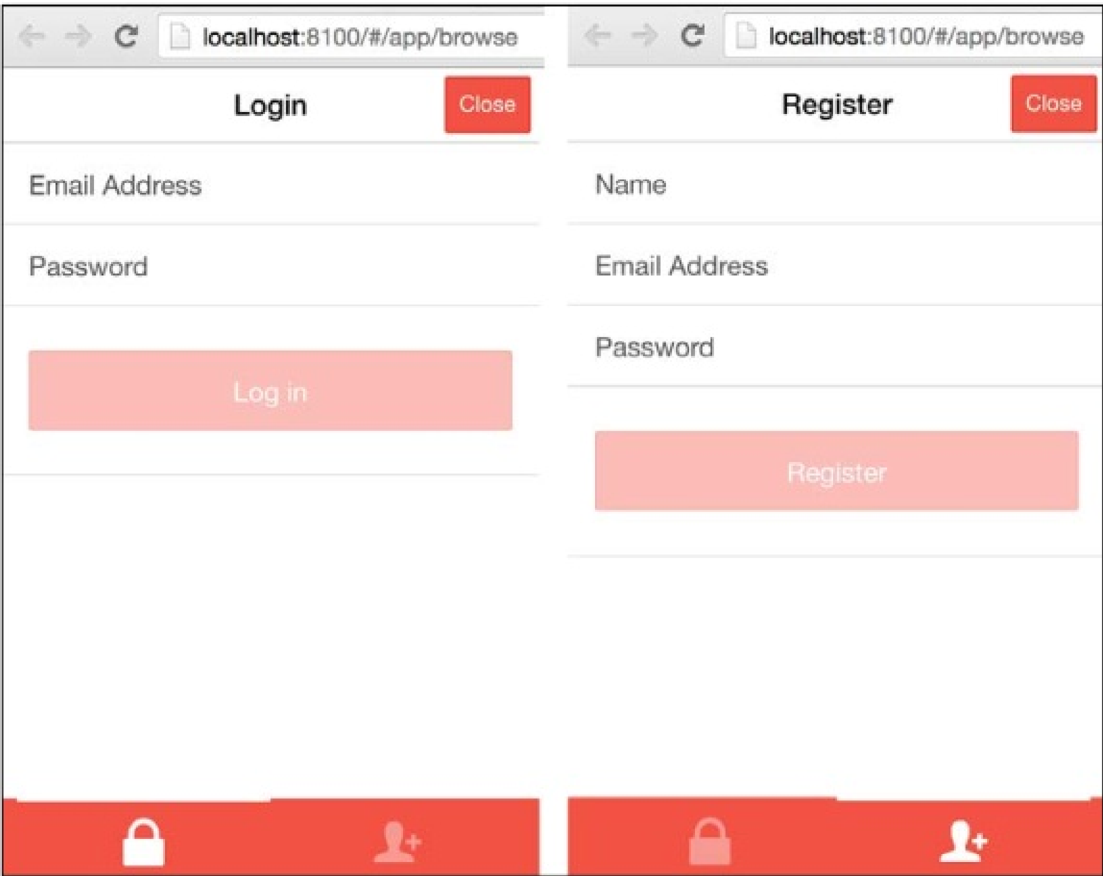
第一个是登录模板， *login.html* ：

```
<ion-modal-view>
<div class="tabs-striped tabs-background-assertive tabs-colorlight">
    <div class="tabs">
        <a class="tab-item" ng-class="{active : viewLogin}" href="javascript:" ng-clic
k="switchTab('login')">
            <i class="icon ion-locked"></i> Login
        </a>
        <a class="tab-item" ng-class="{active : !viewLogin}" href="javascript:" ng-cli
ck="switchTab('register')">
            <i class="icon ion-person-add"></i> Register
        </a>
    </div>
</div>
    <!-- login pane -->
    <ion-pane ng-show="viewLogin">
        <ion-header-bar>
            <h1 class="title">Login</h1>
            <div class="buttons">
                <button class="button button-assertive" ng-click="hide()">Close</butto
n>
            </div>
        </ion-header-bar>
        <ion-content>
            <form>
                <div class="list">
                    <label class="item item-input">
                        <span class="input-label">Email</span>
                        <input type="email" ng-model="user.email">
                    </label>
                    <label class="item item-input">
                        <span class="input-label">Password</span>
                        <input type="password" ng-model="user.password">
                    </label>
                    <label class="item">
                        <button class="button button-block buttonassertive" ng-click="
login()" ng-disabled="!user.email || !user.password" type="submit">Log in</button>
                    </label>
                </div>
            </form>
        </ion-content>
```

```
    </ion-pane>
    <!-- register pane -->
    <ion-pane ng-hide="viewLogin">
        <ion-header-bar>
            <h1 class="title">Register</h1>
            <div class="buttons">
                <button class="button button-assertive" ng-click="hide()">Close</butto
n>
            </div>
        </ion-header-bar>
        <ion-content>
            <form>
                <div class="list">
                    <label class="item item-input">
                        <span class="input-label">Name</span>
                        <input type="text" ng-model="user.name">
                    </label>
                    <label class="item item-input">
                        <span class="input-label">Email  Address </span>
                        <input type="text" ng-model="user.email">
                    </label>
                    <label class="item item-input">
                        <span class="input-label">Password</span>
                        <input type="password" ng-model="user.password">
                    </label>
                    <label class="item">
                        <button class="button button-block buttonassertive" ng-click="
register()" ng-disabled="!user.name || !user.email || !user.password" type="submit" ty
pe="submit">Register</button>
                    </label>
                </div>
            </form>
        </ion-content>
    </ion-pane>
</ion-modal-view>
```

login 模板有一个用在 *Login* 和 *Register* 视图之间切换的标签组件。这个tabs组件不是一个Ionic tab指令，而是一个普通的Tab CSS组件。 点击标签图标的时候，我们将使用 *switchTab* 方法来切换 *viewLogin* 为 *true* 或者 *false* 以在 **Login** 和 **Register** 视图之间进行切换。

完整的登录模板视图效果如下（也可以点击 **Register** 图标来查看注册视图）：



浏览模板（Browse template）

接下来进行的是 *Browse* 模板。这个模板将用来展示书籍列表。打开 *www/templates/browse.html* 更新内容为如下：

```
<ion-view view-title="Browse Books" hide-back-button="true">
    <ion-content>
        <ion-list>
            <div ng-repeat="book in books track by $index" class="row responsive-sm" n
g-if="$index % 2 == 0">
                <ion-item class="col-50" ng-repeat="i in [$index, $index + 1]" ng-if="
books[i] != null" ng-href="#/app/book/{{::books[i]._id}}">
                    <div class="item-thumbnail-left">
                        <img ng-src="{{::book.image}}">
                        <h2>{{::books[i].title}}</h2>
                        <p>{{::books[i].short_description}}</p>
                        <p>
                            <i class="icon ion-star" ng-repeat="i in getNumber(books[i
].rating) track by $index"></i>
                        </p>
                    </div>
                </ion-item>
            </div>
        </ion-list>
    </ion-content>
</ion-view>
```

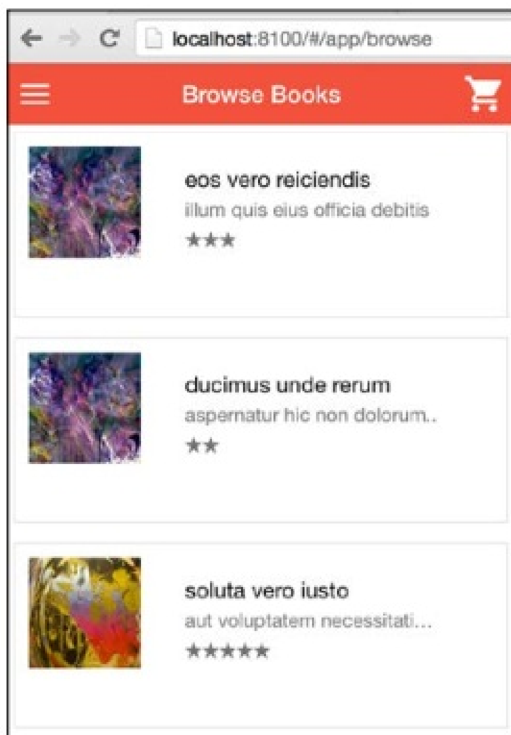
> 之前的模板中，我是用 {{::property}} 来替代。 {{::property}}是AngularJS中单向数据绑定的方法。当模板中使用的属性的值在初始绑定之后后续不会进行更改的时候，单向数据绑定是再理想不过的了 这个应用就使用单向数据绑定就非常完美。值绑定到模板之后，我们就不会在更新他了。这样一来我们可以节省AngularJS对我们变量循环诊断的消耗了。 更多关于单向数据绑定的信息请参考：http://blog.thoughtram.io/angularjs/2014/10/14/exploring-angular-1.3-onetime-bindings.html

在此模板中，我们使用了 *ion-list* 来展示书籍列表。我早先也描述了如何使用Ionic的格子系统来实现。

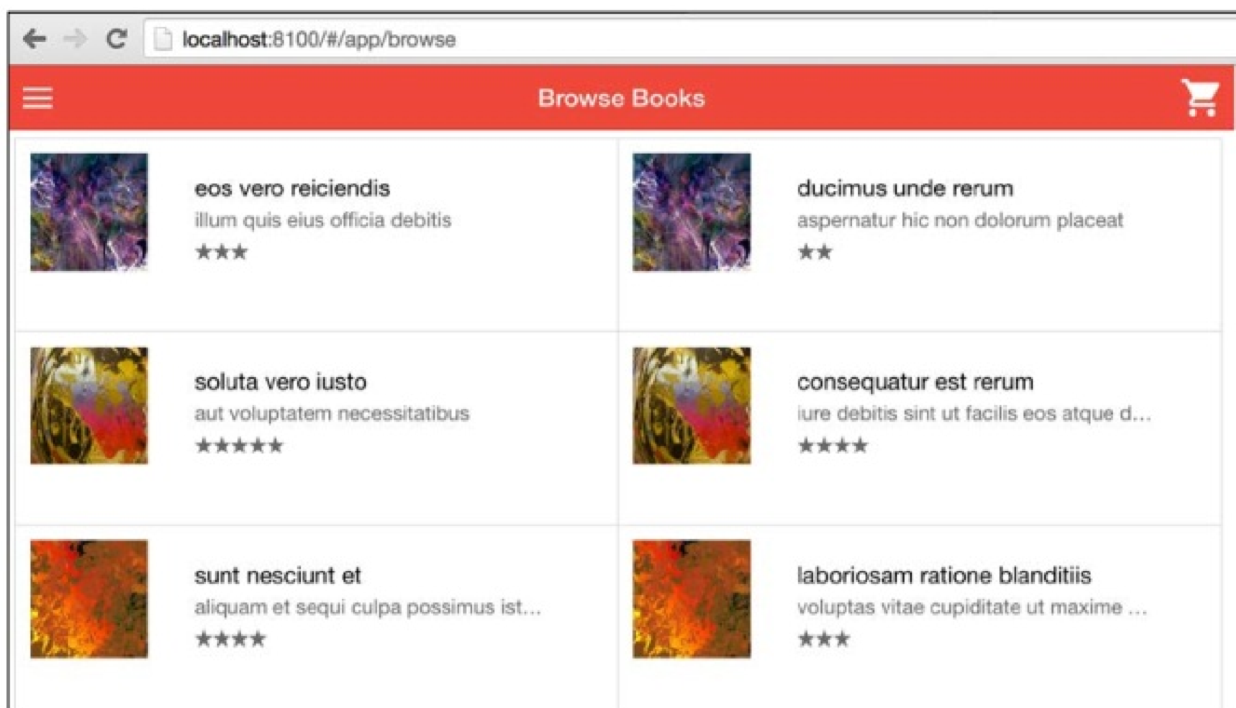
在之前的模板中，我们显示了两行书籍，只是视窗是一个移动设备。我们使用两个循环实现了格子视图。外层循环组成类名为 *item* 的元素，内层循环组成他的条目，类名为 *col-50*

同时，我们给外部循环添加了类 *responsive-sm* 以高速Ionic格子系统，如果设备比较小，那么就渲染为以列就可以了。

移动视窗的渲染效果如下：



桌面版是这样的：



书籍模板（Book template）

当用户在浏览模板里面点击书籍以查看详情的时候，我们会重定向到 */book* 页。在此处我们展示了书籍的详细信息并带有一个 **Add to Cart** 按钮。 以下是 *www/templates/book.html* 的源代码：

```
<ion-view view-title="{{::book.title}}" hide-back-button="true">
    <ion-content>
        <div class="list card">
            <div class="item item-avatar">
                <img ng-src="{{::book.author_image}}">
                <h2>{{::book.title}}</h2>
                <p>{{::book.release_date | date:'yyyy-MM-dd'}}</p>
                <p>{{::book.author}}</p>
            </div>
            <div class="item item-body">
            <img class="full-image" ng-src="{{::book.image}}">
            <p>
            {{::book.long_description}}
            </p>
            <p class="row">
                <label class="col">
                    Rating : <i class="icon ion-star" ngrepeat="i in getNumber(book.ra
ting) track by $index"></i>
                </label>
                <label class="col">
                    Price :
                    <label class="subdued">{{::book.price | currency}} $</label>
                </label>
            </p>
                <button class="button button-assertive buttonblock" ng-click="addToCar
t()">
                <i class="icon ion-checkmark"></i> Add to Cart
                </button>
            </div>
        </div>
    </ion-content>
</ion-view>
```

模板中需要提及的是新增的 **Rating** 部分，此处使用了一个 *ng-repeat* 根据评分值来动态打印星星。我们用到了之前在 *run* 方法里面定义的 *getNumber* 方法。

渲染好的数据模板效果如下：

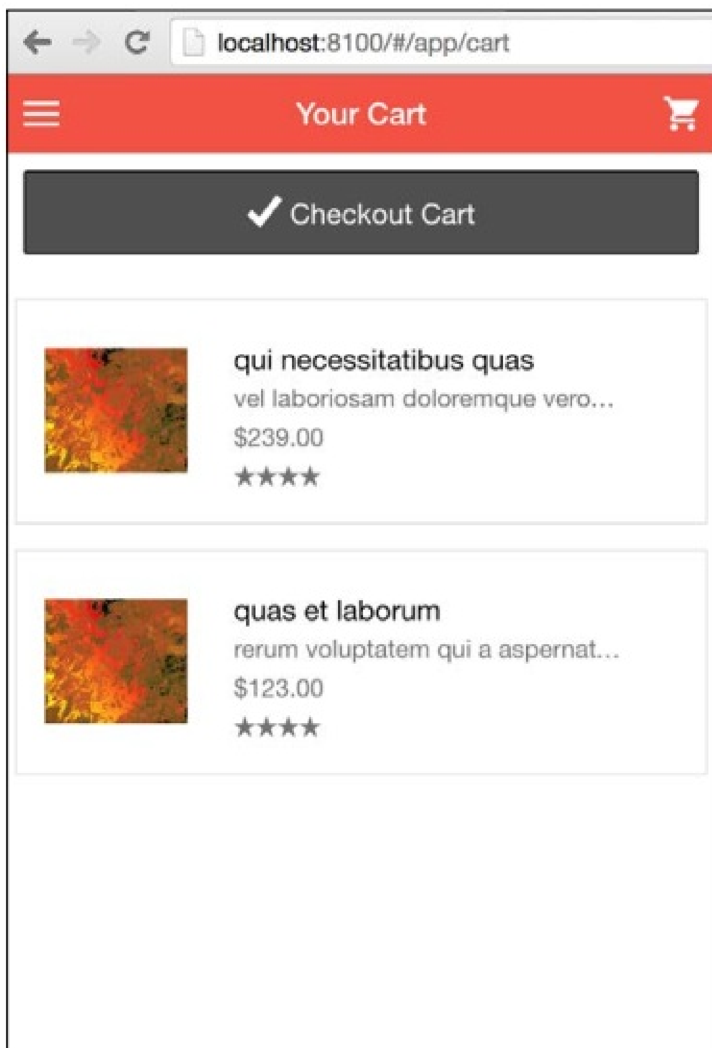


购物车模板（Cart template）

购物车模板跟 **Browse** 模板很想，除了在顶部添加了一个检出（checkout）按钮以及在购物车没有条目的时候的一个倒回信息（fall back message）。*www/templates/cart.html* 源代码如下：

```
<ion-view view-title="Your Cart" cache-view="false" hide-backbutton="true">
    <ion-content>
        <div class="padding">
            <button class="button button-block button-dark" ngshow="books.length > 0"
ng-click="checkout()">
                <i class="icon ion-checkmark"></i> Checkout Cart
            </button>
        </div>
        <ion-list>
            <div ng-repeat="book in books track by $index" class="row responsive-sm" n
g-if="$index % 2 == 0">
                <ion-item class="col-50" ng-repeat="i in [$index, $index + 1]" ng-if="
books[i] != null" ng-href="#/app/book/{{::books[i]._id}}">
                    <div class="item-thumbnail-left">
                        <img ng-src="{{::book.image}}">
                        <h2>{{::books[i].title}}</h2>
                        <p>{{::books[i].short_description}}</p>
                        <p>{{::books[i].price}} $</p>
                        <p>
                        <i class="icon ion-star" ng-repeat="i
                        in getNumber(books[i].rating) track by $index"></i>
                        </p>
                    </div>
                </ion-item>
            </div>
        </ion-list>
        <div class="card" ng-show="books.length == 0">
            <div class="item item-text-wrap text-center">
            <h2>No Books in your cart!</h2>
            <br>
            <a href="#/app/browse">Add a few</a>
            </div>
        </div>
    </ion-content>
</ion-view>
```

渲染后效果如下：



订购模板

最后一个模板是订购模板。（终于！！脖子快要断了！！！）为了让显示方法有所变化，我使用了一个嵌入列表展示订购列表，而不是用一个主详情页。 主详情偏向于在本页上列出所有的订购信息；当用户点击的时候，我们将用户带到另一个页面以显示订购详情。（与 **Browse** 和 **Book** 页面的关系一样）

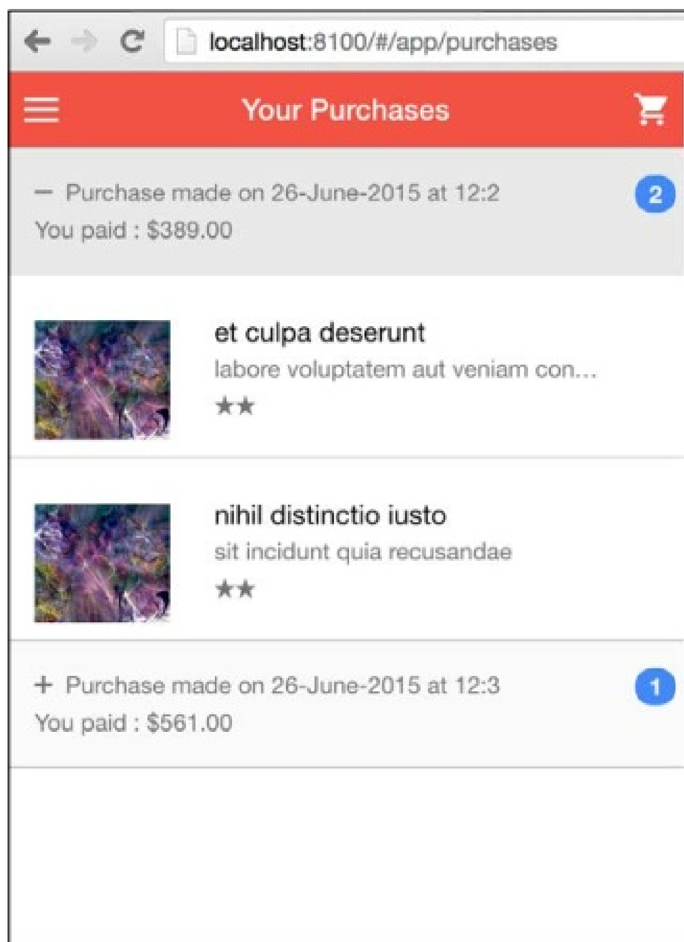
这样，所有订购信息都根据订购时间进行分组，然后作为第一级列表显示。当用户点击组头的时候，我们显示此组内的订购书籍列表。

> 这是一个Accordion组件。更多信息参考：
> http://forum.ionicframework.com/t/expandable-list-in-ionic/3297/2

打开 *www/templates/purchases.html* 更新如下：

```
<ion-view view-title="Your Purchases" cache-view="false" hide-back-button="true">
  <ion-content>
    <ion-list>
      <div ng-repeat="group in groups">
        <ion-item class="item-stable" ngclick="toggleGroup(group)" ng-class="{active:
isGroupShown(group)}">
          <p><i class="icon" ngclass="isGroupShown(group) ? 'ion-minus' : 'ion-plus'">
</i> {{::group.name}}
            <span class="badge badgepositive">{{::group.items.length}}</span></p>
          <p>You paid : {{::group.total | currency}}</p>
        </ion-item>
        <ion-item class="item-accordion" ng-repeat="item in group.items" ng-show="isGr
oupShown(group)" ng-href="#/app/book/{{::item._id}}">
          <div class="item-thumbnail-left">
            <img ng-src="{{::item.image}}">
            <h2>{{::item.title}}</h2>
            <p>{{::item.short_description}}</p>
            <p>
              <i class="icon ion-star" ng-repeat="i in getNumber(item.rating) track by
 $index"></i>
            </p>
          </div>
        </ion-item>
      </div>
    </ion-list>
  </ion-content>
</ion-view>
```

保存文件然后返回浏览器，你将看到：



好了，最后一击：添加一些样式。更新 *www/css/styles.css* 如下：

```
.item-thumbnail-left,
.item-thumbnail-left .item-content {
    min-height: 75px;
}
.ion-android-cart:before {
    font-size: 24px !important;
}
.item-thumbnail-left > img:first-child,
.item-thumbnail-left .item-image,
.item-thumbnail-left .item-content > img:first-child,
.item-thumbnail-left .item-content .item-image {
    top: 27px;
    left: 16px;
    padding-bottom: 10px;
}
.badge.badge-positive {
    position: absolute;
    right: 5px;
}
```

总结

本章中，我们学习了如何利用已有的REST API创建一个Ionic应用。 我们也学习了如何使用基于token的服务器和处理需要认证的路由对比不需要认证的路由。 这个手打范例帮助你巩固了已有的Ionic只是。

下一章中，我们将要学习Cordova插件，以及如何使用ngCordova。

第七章 Cordova与ngCordova

本章中，我们将学习整合设备特别功能到Ionic应用中，例如：网络，电池状态，摄像头等等。首先我们将学习Cordova插件，然后学习ngCordova。
本章涵盖范围：

- 设定一个特定平台的SDK
- 使用Cordova 插件API
- 使用ngCordova
- 测试一些ngCordova插件

设定特定平台SDK

在与设备功能交互之前，我们需要在本机设置设备的操作系统对应的SDK。Ionic只正式支持iOS,Android以及部分Windows phont平台的扩展。 但是，Ionic还是可以用在任何设备上。
以下链接展示了如何在本机上设置移动SDK。 很可惜的是本章不会进行更多的设置。链接如下：

- Android : http://cordova.apache.org/docs/en/5.0.0/guide_platforms_android_index.md.html#Android%20Platform%20Guide
- iOS:http://cordova.apache.org/docs/en/5.0.0/guide_platforms_ios_index.md.html#iOS%20Platform%20Guide
- Windows Phone8 : http://cordova.apache.org/docs/en/5.0.0/guide_platforms_wp8_index.md.html#Windows%20Phone%208%20Platform%20Guide

  > 对于其他OS，参考：
  > http://cordova.apache.org/docs/en/5.0.0/guide_platforms_index.md.html#Platform%20Guides ; 我使用的是Cordova 5.0.0的文档

本书中只对Android与iOS进行设置。其他操作系统也是类似的。
在继续操作之前，我们需要确保设置完成，并且工作正常。

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通:
> https://github.com/learning-ionic/Chapter-7

Android平台设置

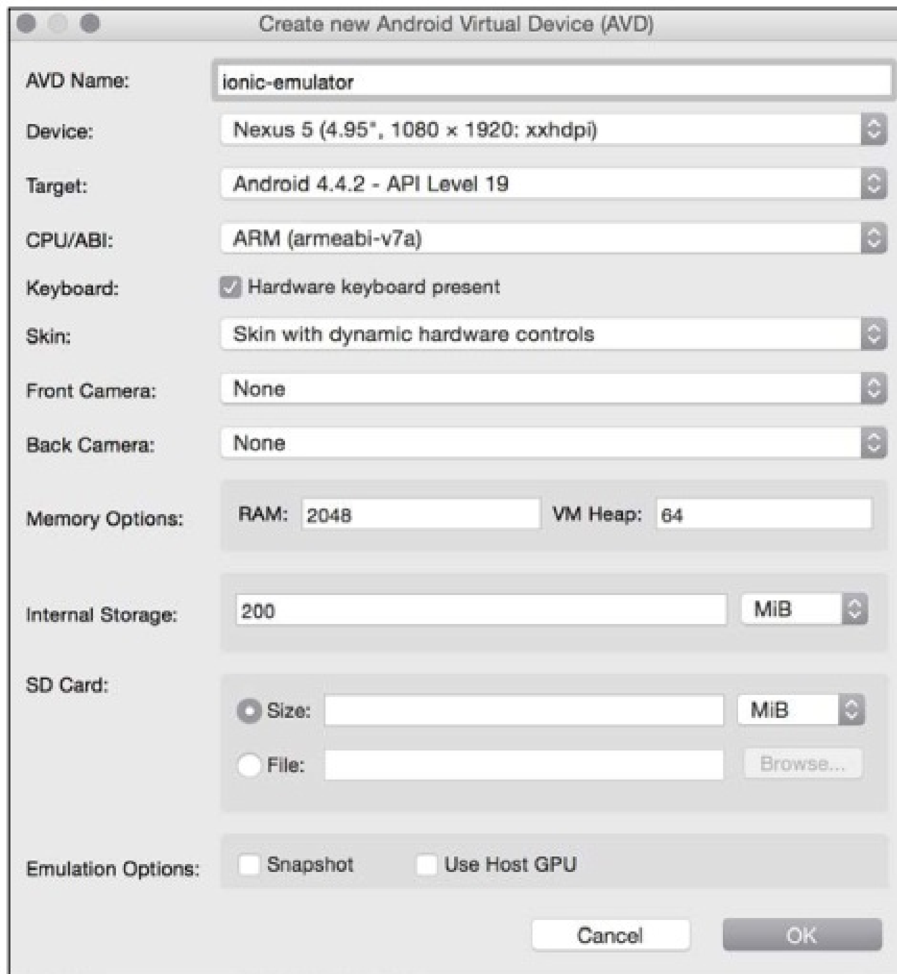
确保安装好了Android的SDK以及Android tools在你的环境变量path中。然后，在任何地方打开命令行/终端，运行：

```
android
```

这个命令将启动Android SDK管理器。先确保你安装了最新版本的Android，或者某个特定的版本。
接下来，运行如下命令：

```
android avd
```

这个命令将启动Android Virtual Device（安卓虚拟设备）管理器。确保至少设置了一个AVD。如果没有的话，点击上面的**Create**按钮创建一个，如下：



iOS平台设置

先确保安装好了Xcode和他所需的工具，同时也要确保全局安装了*ios-sim*和*ios-deploy*：

```
npm install -g ios-sim

npm install -g ios-deploy
```

> iOS设置只能在Apple机器上进行。Windows开发人员不能从Windows上面部署iOS app，因为Xcode只能在iOS上使用。

测试设置

我们看一下如何测试Android和iOS的设置。

测试Android

为测试设置是否成功，我们新建一个Ionic应用，然后使用Android和iOS模拟器进行模拟： 我们将新建一个标签页应用：

```
ionic start -a "Example 27" -i app.example.twentyseven example27 tabs
```

使用*cd*口令进入*example27*文件夹内，运行：

```
ionic serve
```

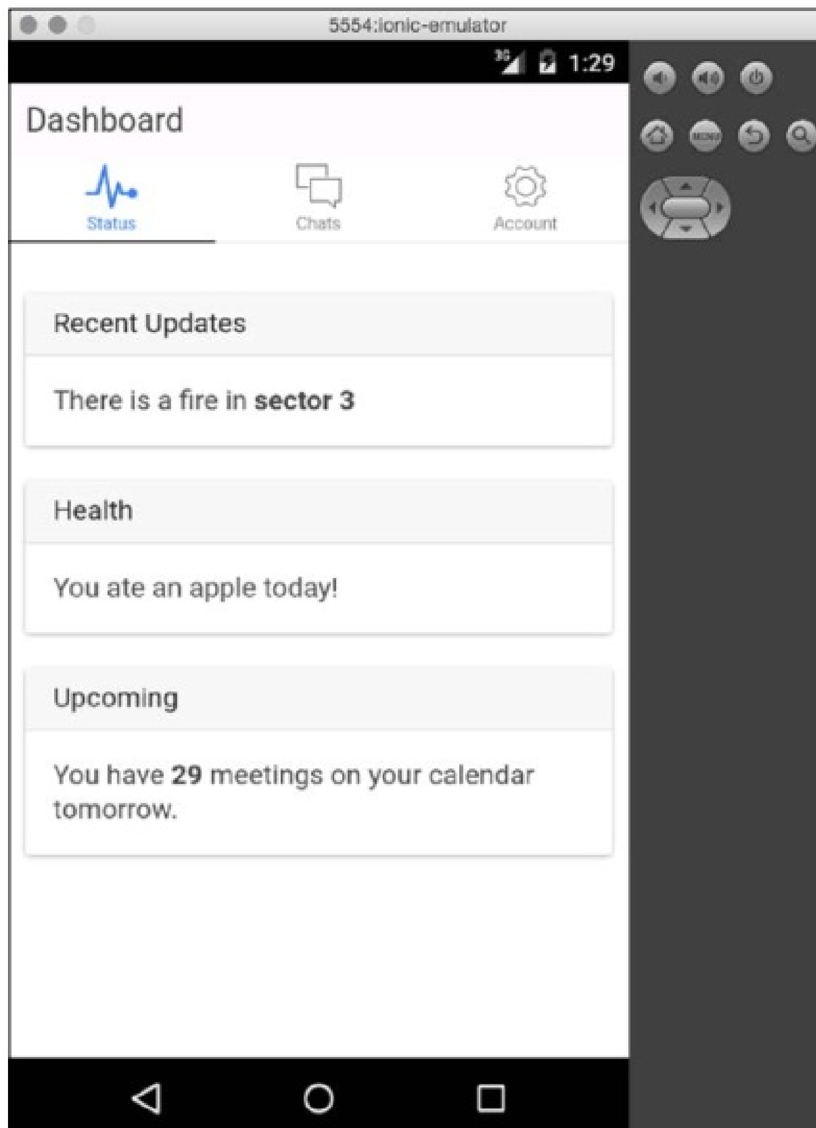
这样，应用就运行起来了。我们就可以通过浏览器访问此应用了。
为了能在Android模拟器中模拟此应用，首先我们需要给项目添加Android平台支持，然后再模拟。
添加Android支持，使用以下命令：

```
ionic platform add android
```

命令运行成功之后，运行此命令：

```
ionic emulate android
```

经过短暂的等待之后，我们可以看到模拟器启动，app部署其中，并且在其中运行：



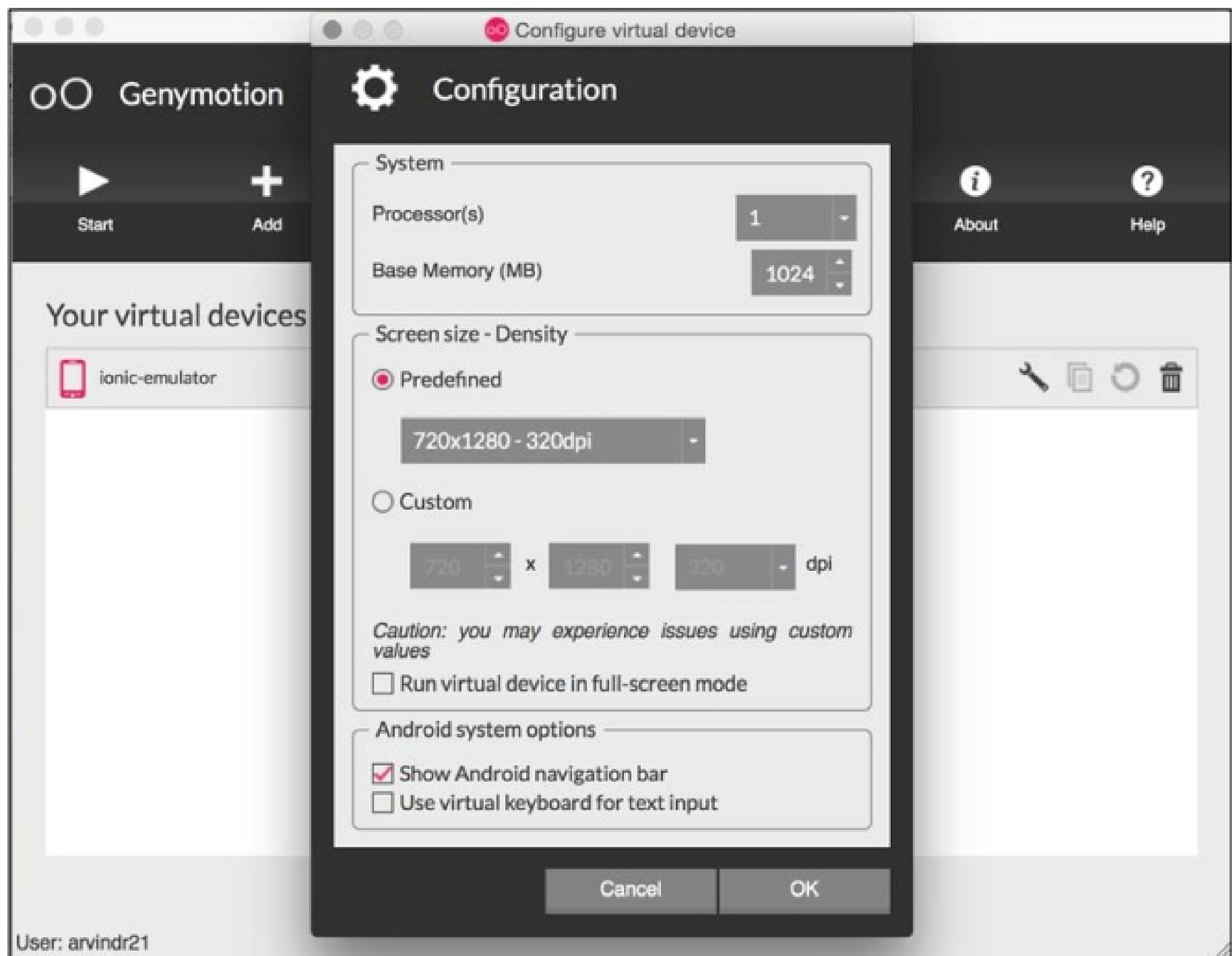
如果你之前用过Android模拟器的话，那么你就应该体会到他有多慢。如果没用过的话，那么我告诉你他真的是好慢。

另一个可选的Android模拟器是Genymotion（(https://www.genymotion.com）。Ionic也很好的与Genymotion整合了。

Genymotion有两个版本，一个免费版和一个商业版。免费版功能较少，仅支持个人使用。

> 可以从此处下载一个Genymotion的副本：https://www.genymotion.com/#!/store

一旦安装好了Genymotion，就可以创建一个你想要的Android SDK的虚拟设备了。以下是我的配置：



接下来我们就可以启动模拟器让他在后台运行了。

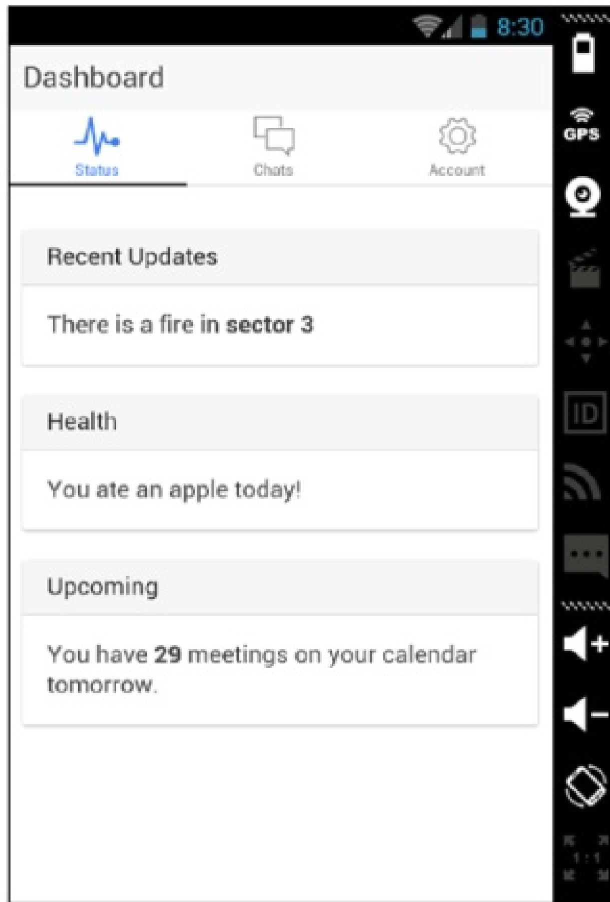
现在，我们的Genymotion模拟器运行起来了，我们就需要告诉Ionic使用Genymotion而不是Android默认的模拟器了，使用如下口令：

```
ionic run android
```

替换：

```
ionic emulate android
```

这个命令将会把app部署到Genymotion模拟器，并且与Android模拟器不同的是，你可以立刻看到效果：



> 一定要确保Genymotion在后台运行。

如果Genymotion对你来说偏贵，那么你也可以简单的连接你的Android移动电话到你的笔记本电脑，然后运行：

```
ionic run android
```

这样，app将会被部署到甚至设备上去。

> 设置Android USB调试，请参考： http://developer.android.com/tools/device.html 之前截屏里的Genymotion是一个私人版的，因为我没有买授权。 开发期间我都是使用iOS模拟器连接我的Android手机的。一旦完成开发，我从在线测试服务订购设备使用，然后在上面进行测试。 如果你在连接Android手机与电脑的时候出现问题，请先检查一下时候可以在命令行/终端里面运行行*adb*命令，且命令可以列出你的设备。 更多关于Android Debug Bridge（ADB）的信息，请参考：http://developer.android.com/tools/help/adb.html

以上是测试Android app的不同方式。

测试iOS

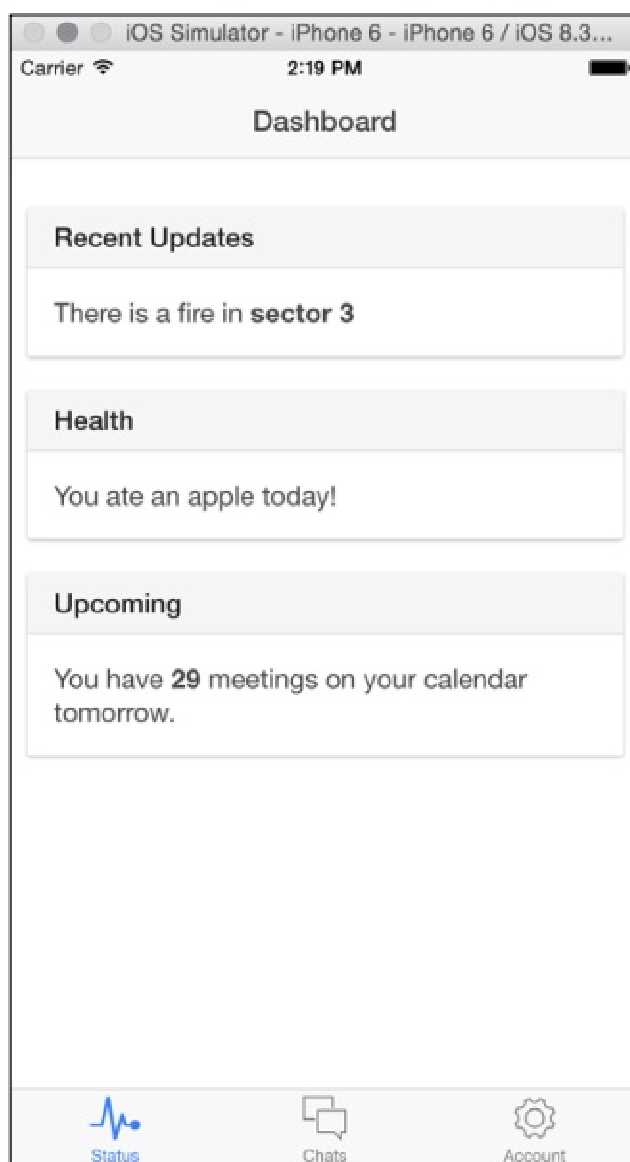
要测试iOS，首先要添加iOS支持，然后进行模拟。
运行：

```
ionic platform add ios
```

然后：

```
ionic emulate ios
```

然后你可以看到默认模拟器运行，最后app出现了：



可以使用以下命令部署到Apple设备：

```
ionic run ios
```

深入之前要确保你可以模拟app。

> 名词参考：
>
> - dot notation：点式标记 eg:org.apache.cordova.camera，
> - hyphenated notation：连字标记 eg:org-apache-cordova-camera

开始使用Cordova插件

参考Cordova文档：

> "A plugin is a package of injected code that allows the Cordova web view within which the app renders to communicate with the native platform on which it runs. Plugins provide access to device and platform functionality that is ordinarily unavailable to web-based apps. All the main Cordova API features are implemented as plugins, and many others are available that enable features such as bar code scanners, NFC communication, or to tailor calendar interfaces." 插件是一个注入包，他允许渲染app的web view与他运行的本地平台进行通讯。 插件提供了设备和平台功能的访问接口，这些是网页应用访问不了的。 所有的Cordova主体API都实现为插件，也有一些其他可用的接口，例如二维码扫描，NFC通讯以及tailor calendar。

换句话说，Cordova插件是你的设备功能的窗口。Cordova/Phonegap团队以及为基本上所有的设备功能创建了可用的插件。 也有社区贡献的插件，他们自定义封装了很多设备特定的功能。

> 可以在此处查找已有的插件：http://plugins.cordova.io/

在本章教程期间，我们将探索一些插件。

> 注意Cordova插件正在迁移到NPM。此书发布的时候；所有的Cordova将会完成到NPM的迁移。更多信息，参考：https://cordova.apache.org/announcements/2015/04/21/plugins-releaseand-move-to-npm.html

为调整之前的变更，作为一个开发人员你这边不需要做任何变更，除了使用Cordova CLI（版本大于等于5.0.0）来添加插件。 这样将会从合适的注册点下载对应的插件。

> Ionic团队已经合并了一个pull请求：将dot notation改为hyphenated notation。参考：https://github.com/driftyco/ionic-cli/pull/409

由于我们的关注点是Ionic开发，我们将使用Ionic CLI来添加插件。他本身就是调用Cordova CLI的。

Ionic插件API

在使用插件的过程中，主要有4个用到的命令。

添加插件

这个CLI命令用来给项目添加一个新的插件，例如：

```
ionic plugin add org.apache.cordova.camera
```

也可以这样：

```
ionic plugin add cordova-plugin-camera
```

移除插件

这个CLI命令用来从项目移除一个插件的，例如：

```
ionic plugin rm org.apache.cordova.camera
```

也可以这样：

```
ionic plugin rm cordova-plugin-camera
```

列出添加的插件

此CLI命令是用来列出所有项目里的插件的，例如：

```
ionic plugin ls
```

查找插件

此CLI命令是用在命令行中搜索插件的，例如：

```
ionic plugin search scanner barcode
```

为测试以上命令，新建一个项目然后逐个测试一下：

```
ionic start -a "Example 28" -i app.example.twentyeight example28 blank
```

> 创建空白模板的时候，我们会下载和设置一下几个插件：
>
> - cordova-plugin-device
> - cordova-plugin-console
> - cordova-plugin-whitelist
> - cordova-plugin-splashscreen
> - com.ionic.keyboard

使用*cd*命令进入*example28*文件夹内运行如下命令以备测试：

```
ionic serve
```

我们先来搜索电池状态插件，然后将他添加到项目。关闭服务然后运行：

```
ionic plugin search battery status
```

编写本章的时候，看到的是如下的结果：

```
→  example28  ionic plugin search battery status
Updated the hooks directory to have execute permissions
running cordova plugin search battery status
npm http GET http://registry.cordova.io/-/all/since?stale=update_after&s
tartkey=1433066172224
npm http 200 http://registry.cordova.io/-/all/since?stale=update_after&s
tartkey=1433066172224
com.blueshift.cordova.battery - Battery
org.apache.cordova.battery-status - Cordova Battery Plugin
→  example28
```

运行此命令，有可能只找得到hyphenated版本，也许都可以找得到。
根据搜索到的插件名字，将它添加到项目中。
所以，在我们案例中，我将运行如下命令以将电池状态的插件添加到项目：

```
ionic plugin add org.apache.cordova.battery-status
```

这样电池状态插件（https://github.com/apache/cordovaplugin-battery-status）将会添加到当前项目。
此时，再次运行上面的命令时，将会看到：

```
→  example28  ionic plugin add org.apache.cordova.battery-status
Updated the hooks directory to have execute permissions
running cordova plugin add org.apache.cordova.battery-status
WARNING: org.apache.cordova.battery-status has been renamed to cordova-plugin-battery-status. You
 may not be getting the latest version! We suggest you `cordova plugin rm org.apache.cordova.batt
ery-status` and `cordova plugin add cordova-plugin-battery-status`.
Fetching plugin "org.apache.cordova.battery-status" via cordova plugins registry
npm http GET http://registry.cordova.io/org.apache.cordova.battery-status
npm http 200 http://registry.cordova.io/org.apache.cordova.battery-status
npm http GET http://cordova.iriscouch.com/registry/_design/app/_rewrite/org.apache.cordova.batter
y-status/-/org.apache.cordova.battery-status-0.2.12.tgz
npm http 200 http://cordova.iriscouch.com/registry/_design/app/_rewrite/org.apache.cordova.batter
y-status/-/org.apache.cordova.battery-status-0.2.12.tgz
Saving plugin to package.json file
Adding since there was no existingPlugin
→  example28
```

CLI打印出来一个警告告诉你插件已被重命名，下载的插件可能不是最新的。
那么，我们来使用最新版吧。但是，在我们使用连字版本之前，我们需要移除已经添加插件，如下：

```
ionic plugin rm org.apache.cordova.battery-status
```

添加连字命名方式的插件：

```
cordova plugin add cordova-plugin-battery-status
```

想要查看我们安装的所有插件的话，运行如下命令：

```
ionic plugin ls
```

然后你会看到以下：

```
→  example28  ionic plugin ls
Updated the hooks directory to have execute permissions
com.ionic.keyboard 1.0.4 "Keyboard"
cordova-plugin-battery-status 1.1.0 "Battery"
cordova-plugin-console 1.0.1 "Console"
cordova-plugin-device 1.0.1 "Device"
cordova-plugin-splashscreen 2.1.0 "Splashscreen"
cordova-plugin-whitelist 1.0.0 "Whitelist"
```

之前说过，*com.ionic.keyboard*是用的连字标记法，其他的都是用的点式标记法。 当你运行这个命令的时候，应该只能看到连字命名标记。
在继续测试电池状态插件之前，我们需要在代码中用别的方式来使用。打开*www/js/app.js*。
在*run*方法里，快到*ionicPlatform.ready*回调的地方，添加以下代码：

```
alert(device.model);
window.addEventListener("batterystatus", onBatteryStatus,false);
function onBatteryStatus(info) {
    // Handle the online event
    alert("Level: " + info.level + " isPlugged: " + info.isPlugged);
}
```

我们添加了一个警告窗口展示设备模型（需要用到*cordova-plugin-deveice*插件），然后添加了一个电池状态变更的事件监听（需要用到*cordova-plugin-battery-status*）。
此时，运行：

```
ionic serve
```

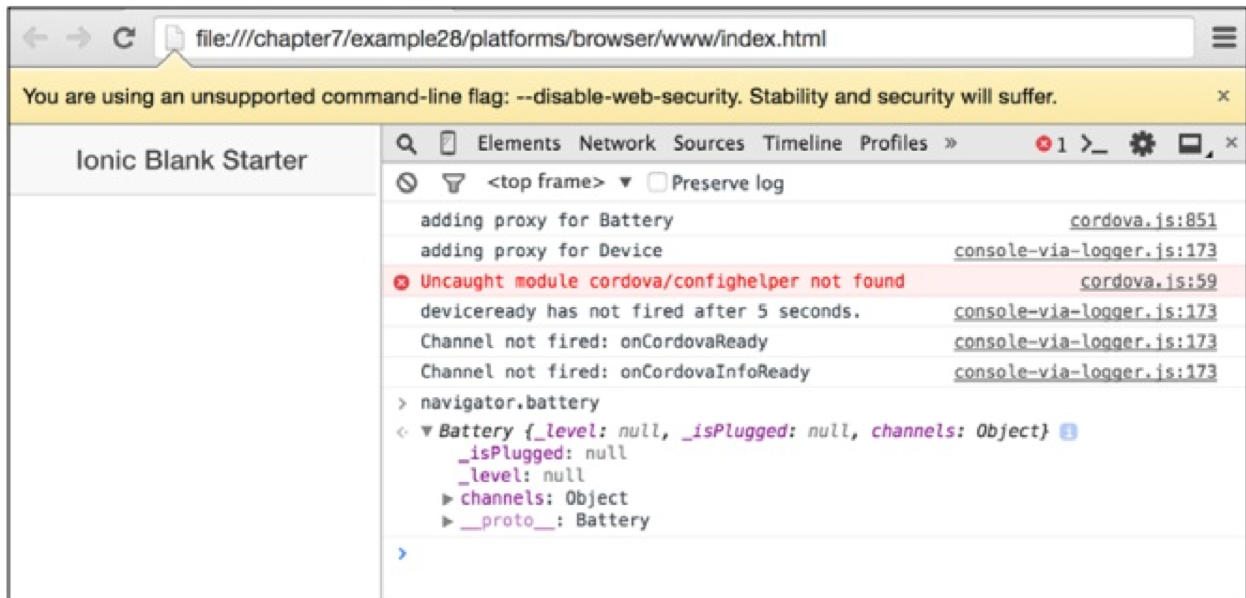
你将看到并没有警告窗口弹出来，然后，当你打开开发者工具的时候，会发现报错了：*device is not defined*
这是因为我们不能直接在浏览器中运行这些插件；他们需要一些环境去执行，例如Android，iOS，或者是他本身（移动系统本身）的浏览器。
是的，浏览器是一个单独的平台，像Android，iOS一样，他用来运行*cordova.js*。*cordova.js*是连接JavScript和设备特定语言之间缝隙的桥梁。 根据你使用的插件的类型的不同，你可以选择在浏览器中对部分插件进行模拟。
我们用这个来试试设备和电池功能。添加一个浏览器平台：

```
ionic platform add browser
```

现在在浏览器平台运行此app，如下：

```
ionic run browser
```

此时将启动一个新的当前默认浏览器的实例，在其中运行此app。现在，就可以看到*device.model*的警告框了。 此时，打开开发者工具，可以看到：



此时，如果在浏览器控制台中运行*navigator.battery*的时候，会发现*battery*对象里面都是*null*属性。

为更好的测试测试此app（和插件），我们需要添加一个Android平台或者iOS平台：

```
ionic platform add android
```

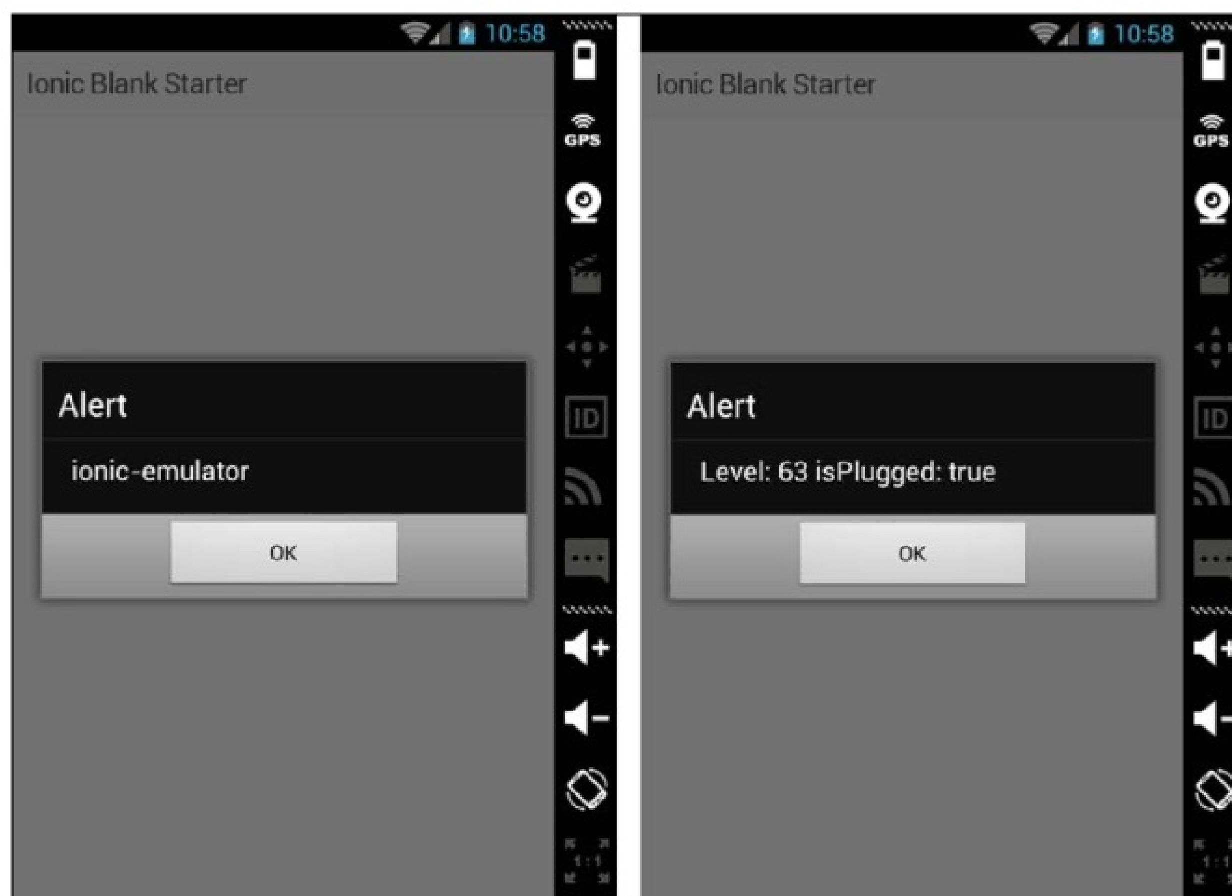
或者：

```
ionic platform add ios
```

然后执行以下任意一个命令：

- ionic emulate android
- ionic emulate ios
- ionic run android
- ionic run ios

然后你就可以看到正确的显示信息了：



现在，你已经知道怎样给Ionic项目添加Cordova插件以及如何对他们进行测试了。 下一步分钟，我们将学习ngCordova和其他一些插件。

> 以上截屏来自我的个人版Genymotion。此图仅供展示之用。

Cordova白名单（whitelist）插件

在开始学习ngCordova之前，我们将花点时间熟悉一个关键的Cordova插件 - 白名单插件 https://github.com/apache/cordova-plugin-whitelist

Cordova文档里面是这样描述白名单插件的：

> "Domain whitelisting is a security model that controls access to external domains over which your application has no control. Cordova provides a confgurable security policy to defne which external sites may be accessed." 领域白名单是一个安全模型，用来控制外部域名的访问，在此之上你的应用没有控制权。 Cordova提供了一个可配置的安全策略供配置哪些外部网站可以访问。

所以当你在使用其他资源内容，想要更多控制权的时候，你就会用得到白名单插件。你也许已经注意到了，咱们的Ionic项目已经添加了白名单插件。

译者：听起来与*ActionScript 3*的*SecurtiyDomain.allowDomain*差不多

如果Ionic项目或者Cordova项目没有添加此插件的，只要运行如下命令就可以添加了：

```
ionic plugin add cordova-plugin-whitelist
```

一旦插件添加完成，就可以去更新*config.xml*里面的导航白名单-也就是你的app允许在webview里面访问的连接。
你可以这样添加让app的webview里面可以访问*example.com*：

```
<allow-navigation href="http://example.com/*" />
```

如果你想要允许webview访问任何网站的话：

```
<allow-navigation href="http://*/*" />
<allow-navigation href="https://*/*" />
<allow-navigation href="data:*" />
```

你也可以添加一个Intent白名单，在此处可以指定允许在设备上浏览的链接列表。例如，从我们的app里面打开SMS app：

```
<allow-intent href="sms:*" />
```

或者一个简单的网页：

```
<allow-intent href="https://*/*" />
```

也可以通过插件在设备上增加Content Security Policy（CSP） http://content-securitypolicy.com/ 。你所要做的只是在*www/index.html*的*meta*标签里面添加如下内容：

```
<!-- Allow XHRs via https only -->
<meta http-equiv="Content-Security-Policy" content="default-src 'self' https:">
```

这是针对白名单插件的一个快速浏览，白名单插件适用于：

- Android 4.0.0及以上版本
- iOS 4.0.0及以上版本

> 记住，要添加此插件并且配置好才能访问外部链接。在第六章 创建一个书店应用里面，确保白名单插件正确设置；否则，当你将app部署到设备上的时候，app运行会有问题。

ngCordova

之前的案例中，我们整合了一些插件，并且使用了他们的JavaScript API与他们进行交互。你可能发现了，所有插件都处于同一个全局命名空间。 与AngularJS的依赖注入哲学不同的是，Cordova插件处于用以全局命名空间之下可以从任何地方进行访问。当使用依赖注入理念搭建的应用在测试的时候，可能会带来麻烦。
因此，Ionic团队封装了Cordova插件，由此你就可以将这些功能作为服务进行注入。在之前的案例中，我们注入一个*$cordovaDevice*，然后使用*$cordovaDevice.getModel*方法来替换*device.model*。
ngCordova库不是为Ionic特定的；他也可以用在任何使用AngularJS制作的Cordova app中。
本章编写的时候，ngCordova已有71个插件。
现在，我们测试驱动一些ngCordova插件。

设置ngCordova

使用ngCordova之前，我们需要下载并添加他为依赖。新建一个空白模板项目来测试一下：

```
ionic start -a "Example 29" -i app.example.twentynone example29 blank
```

接下来，添加ngCordova作为项目依赖。使用*cd*命令进入*example29*：

```
bower install ngCordova --save
```

为验证ngCordova在项目中是否添加正确，导航至*www/lib*文件夹，你将会看到一个名为ngCordova的文件夹；在*dist*文件夹内，你可以找到一个文件名为*ng-cordova.min.js*。
接下来，添加此文件的引用，然后将ngCordova作为依赖注入项目。
打开*www/index.html*，加上：

```
<!-- ngCordova -->
<script src="lib/ngCordova/dist/ng-cordova.min.js"></script>
```

> *ngCordova脚本应该放在ionic.bundle.js之后，cordova.js之前。如果顺序错误，控制台会有错误信息输出。*

接下来，我们需要在我们的模组中添加ngCordova依赖。打开*www/js/app.js*，更新Angular模组声明：

```
angular.module('starter', ['ionic', 'ngCordova'])
```

由于*cordova-plugin-device*已经预装好了，我们现在可以使用*$cordovaDevice*服务了。
如下更新*www/js/app.js*中的*run*方法：

```
.run(function($ionicPlatform, $cordovaDevice) {
    $ionicPlatform.ready(function() {
        // Hide the accessory bar by default (remove this to show the accessory bar ab
ove the keyboard
        // for form inputs)
        if (window.cordova && window.cordova.plugins.Keyboard) {
            cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
        }
        if (window.StatusBar) {
            StatusBar.styleDefault();
        }
        alert('Platform : ' + $cordovaDevice.getPlatform() + '\nModel: ' + $cordovaDev
ice.getModel());
    });
})
```

我们就可以看到警告弹出设备平台和模组信息了。

> *任何处理插件相关的代码必须放在$ionicPlatform.ready中*

Legend 说明

从今往后，当我说，“给Ionic app添加一个平台”的时候，意味着你应该运行：

```
ionic platform add android
```

也可以：

```
ionic platform add ios
```

当我说，“为Ionic app添加ngCordova支持”的时候，意味着你应该运行：

```
bower install ngCordova --save
```

接下来，跟早先一样，在*www/index.html*中加入*ng-cordova.min.js*。最后将ngCordova添加到AngularJS模组作为依赖。
当我说，“模拟Ionic app”的时候，意味着你应该运行：

```
ionic emulate android
```

或者：

```
ionic emulate ios
```

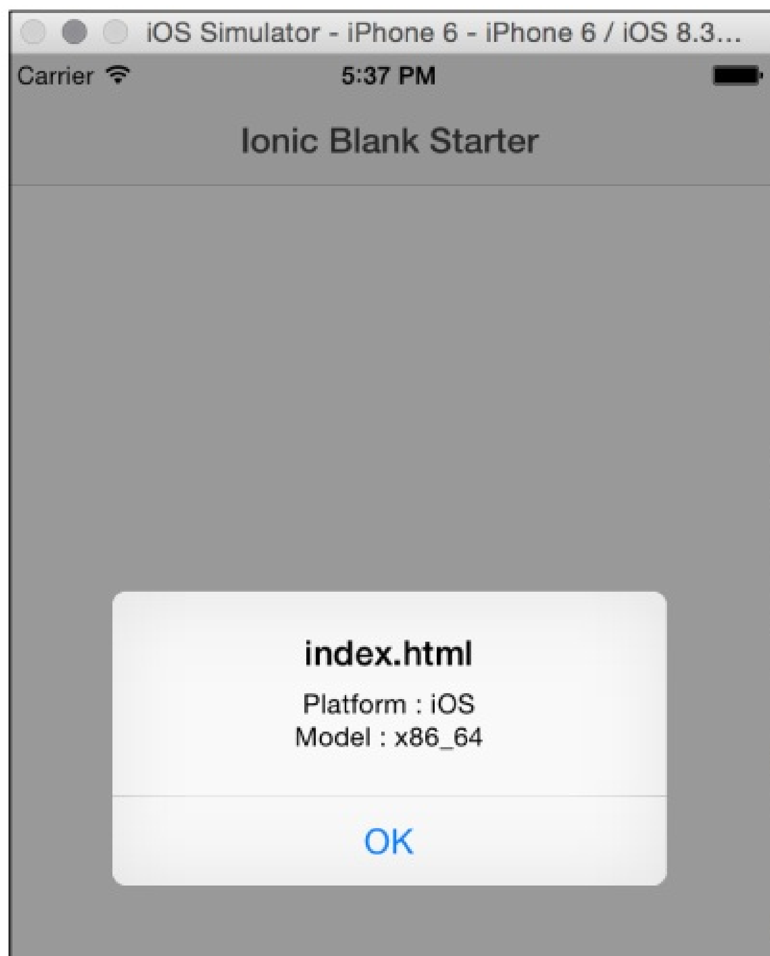
最后，当我说“运行Ionic app”的时候，意味着你应该运行：

```
ionic run android
```

或者：

```
ionic run ios
```

现在，给*example29* app添加平台并进行模拟的时候。可以看到：



这是一个使用ngCordova的端对端的范例。下一部分，我们将通过ngCordova服务来使用一些Cordova插件。

每个插件我都会创建一个新的项目，这样你后续就可以轻松的进行参考。如果你跟随我的脚步进行练习的话，最好别这样做；所有插件用在一个项目里就好了。

$cordovaToast

第一个学习的插件是toast插件。这个插件弹出一个展示文本而不会阻断app的用户交互。
新建一个空白模板项目：

```
ionic start -a "Example 30" -i app.example.thirty example30 blank
```

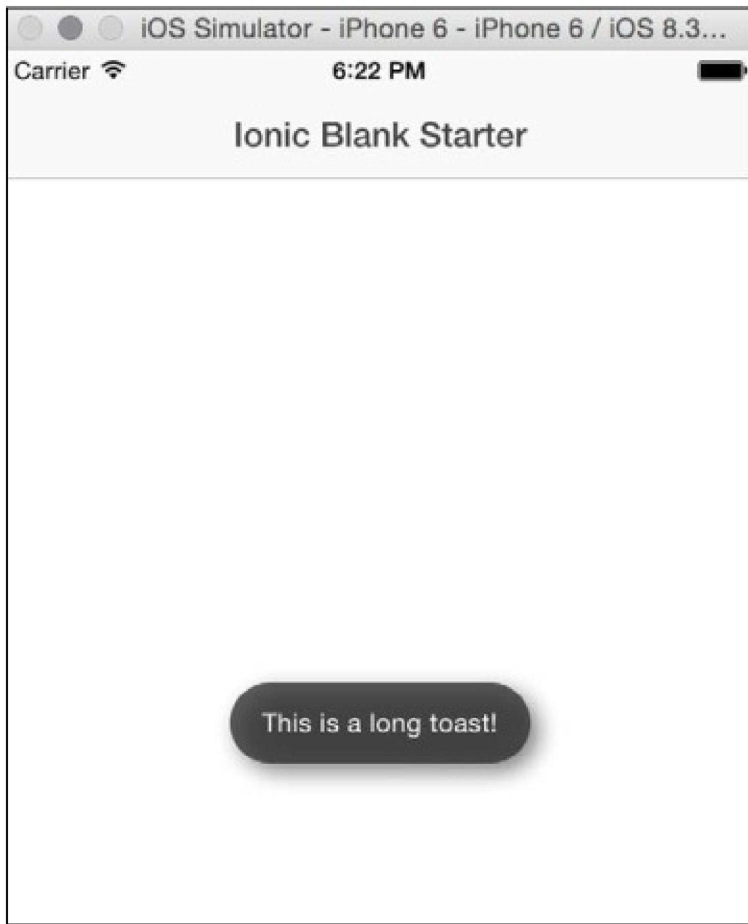
接下来，给项目添加ngCordova支持。想要使用toast API的话，那么得先将toast插件添加到项目里：

```
ionic plugin add https://github.com/EddyVerbruggen/Toast-PhoneGapPlugin.git
```

现在，我们给每个需要用到的插件分别创建一个控制器，而不是去*run*方法中处理。这样，当你回顾参考的时候就会比较简单。
打开*www/index.html*在*body*标签上添加*ng-controller="ToastCtrl"*。然后在*www/js/app.js*中的*run*方法下面，添加控制器定义代码：

```
.controller('ToastCtrl', ['$ionicPlatform', '$cordovaToast',function($ionicPlatform, $
cordovaToast) {
    $ionicPlatform.ready(function() {
        $cordovaToast
            .show('This is a long toast!', 'long', 'center')
            .then(function(success) {
            // success
            }, function(error) {
            // error
            });
    });
}])
```

接着，给Ionic App添加一个平台然后模拟之。效果图如下：



在本范例中，我们将使用仅可能小版本的API方法。在学完每个插件之后，我们提供他们的API链接；你可以查看他们支持的其他方法。

> 更多信息，请访问：http://ngcordova.com/docs/plugins/toast/

$cordovaDialogs

接着学习的是对话框插件。他会触发警告，确认以及提醒窗。
新建一个空白模板项目：

```
ionic start example31 blank
```

接下来，给项目添加ngCordova支持。然后使用以下命令给项目添加对话框插件：

```
ionic plugin add cordova-plugin-dialogs
```

我们将创建一个对话框控制器。打开*www/index.html*在*body*标签处添加*ng-controller="DialogsCtrl"*。
本范例中，我们将显示一个提醒框供用户输入文本。用户在输入文本的时候，我们将文本输

出到屏幕上。我们将*www/index.html*的body部分更新如下：

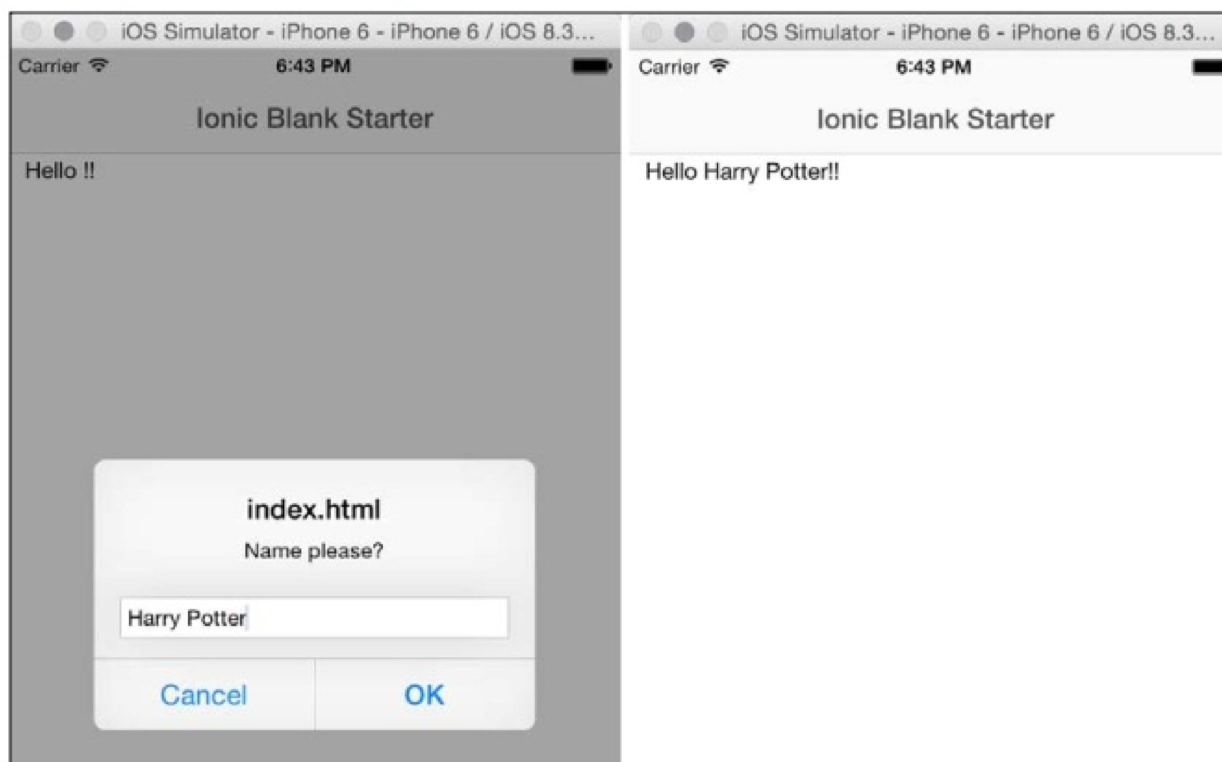
```
<body ng-app="starter" ng-controller="DialogsCtrl">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Ionic Blank Starter</h1>
        </ion-header-bar>
        <ion-content>
            <span class="padding">Hello {{name}}!!</span>
        </ion-content>
    </ion-pane>
</body>
```

在*www/js/app.js*中添加*DialogsCtrl*：

```
.controller('DialogsCtrl', ['$ionicPlatform', '$scope','$cordovaDialogs', function ($i
onicPlatform, $scope,$cordovaDialogs) {
    $ionicPlatform.ready(function () {

        $cordovaDialogs.prompt('Name please?', 'Identity',    ['Cancel', 'OK'], 'Harry
 Potter')
            .then(function (result) {
                if (result.buttonIndex == 2) {
                    $scope.name = result.input1;
                }
            });
    });
}])
```

接下来给Ionic App添加一个平台然后进行模拟。效果图如下：



> 更多信息参考： http://ngcordova.com/docs/plugins/dialogs/

$cordovaFlashlight

下一个学习的是一个工具插件。这个插件用户开关设备上的手电筒。这个插件不能在模拟器上测试。所以你得找个设备来测试这个插件。
新建一个空白模板项目：

```
ionic start -a "Example 32" -i app.example.thirtytwo example32 blank
```

接下来，给项目添加ngCordova支持。运行如下命令添加手电筒插件到项目：

```
ionic plugin add https://github.com/EddyVerbruggen/FlashlightPhoneGap-Plugin.git
```

打开*www/index.html*，在*body*标签处添加*ng-controller="FlashlightCtrl"*。
本例中，我们使用*ion-toggle*指令给用户展示一个开关，然后根据他的状态，对设备上的手电筒进行开与关操作。由此，更新*www/index.html*如下：

```
<body ng-app="starter" ng-controller="FlashlightCtrl">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Ionic Blank Starter</h1>
        </ion-header-bar>
        <ion-content>
            <ion-list>
                <ion-item>
                    <ion-toggle ng-disabled="notSupported" ng-model="torch" ng-change=
"toggleTorch()">
                    Torch
                    </ion-toggle>
                </ion-item>
            </ion-list>
        </ion-content>
    </ion-pane>
</body>
```

在*www/js/app.js*的*run*方法后面添加*FlashlightCtrl*：

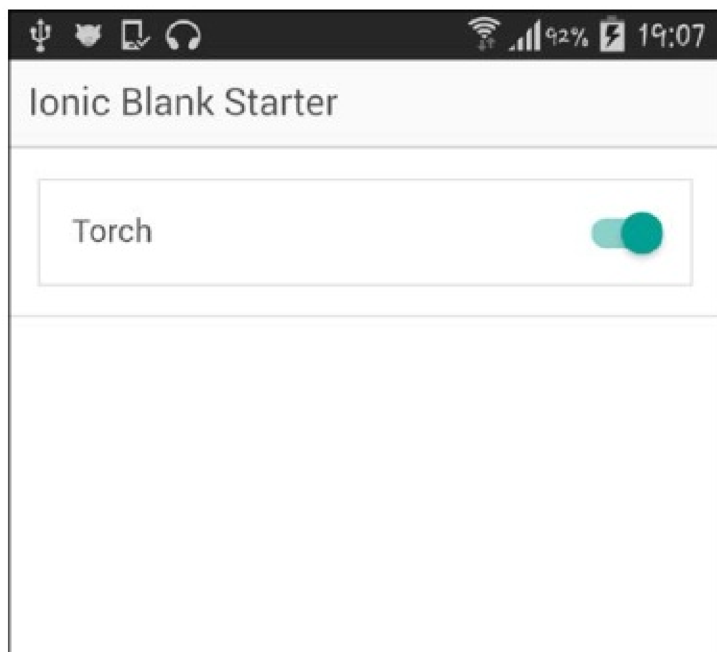
```
.controller('FlashlightCtrl', ['$scope', '$ionicPlatform','$cordovaFlashlight', functi
on($scope, $ionicPlatform,$cordovaFlashlight) {
    $scope.notSupported = true;

    $ionicPlatform.ready(function() {
        $cordovaFlashlight.available().then(function(availability)
        {
            // availability = true || false
            $scope.notSupported = !availability;
        });
        $scope.toggleTorch = function() {
            if ($scope.notSupported) return;

            $cordovaFlashlight.toggle()
                .then(function(success) { /* success */ },
                function(error) { /* error */ });
        }
    });
}])
```

我们先检查插件是否可用。如果可用，我们激活开关；否则，保持开关的禁用状态。
然后在用户切换开关的时候，我们调用*toggleTorch*方法，这样就可以切换手电筒的开关状态了。

如果你在设备上运行app的时候，将会看到如下效果：



如果想要验证开关是否会禁用，可以在模拟器中模拟此app。

> 更多信息参考：http://ngcordova.com/docs/plugins/flashlight/

$cordovaLocalNotification

接下来学习的是Notification（通知）插件。这个插件主要是用来通知或者提醒用户App相关的活动。有时候，当后台运行一些活动的时候也会显示通知-例如，上传大文件的时候。
新建一个空白模板App：

```
ionic start -a "Example 33" -i app.example.thirtythree example33 blank
```

接下来，给项目添加ngCordova支持。运行如下命令添加通知插件到项目：

```
ionic plugin add de.appplant.cordova.plugin.local-notification
```

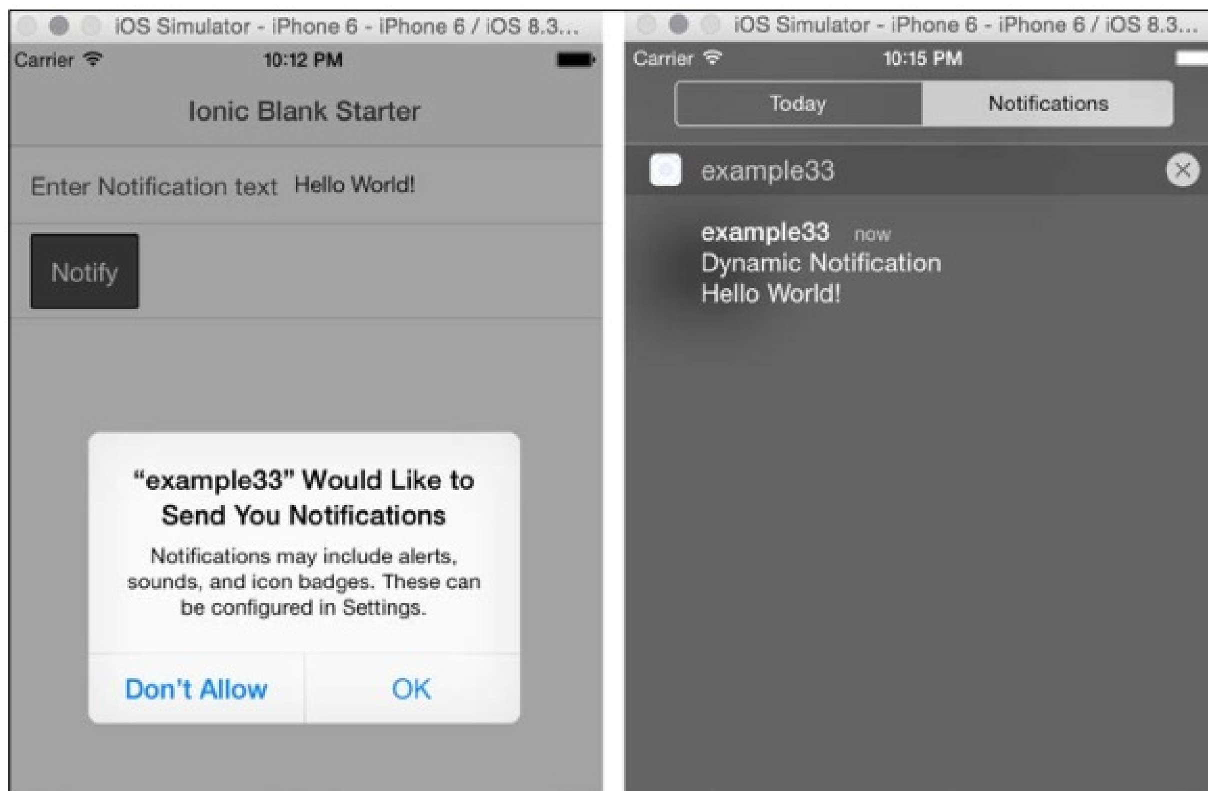
在本例中，我们将在按下按钮的时候触发一个通知，然后展示用户在文本框中输入的文本。因此，我们需要添加一个名为*NotifCtrl*的控制器和一个文本框，一个按钮。
更新*www/index.html*的*body*部分为以下代码：

```
<body ng-app="starter" ng-controller="NotifCtrl">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Ionic Blank Starter</h1>
        </ion-header-bar>
        <ion-content>
            <div class="list">
                <label class="item item-input">
                    <span class="input-label">Enter Notification text</span>
                    <input type="text" ng-model="notifText">
                </label>
                <label class="item item-input">
                    <button class="button button-dark" ng-click="triggerNotification()
">
                    Notify
                    </button>
                </label>
            </div>
        </ion-content>
    </ion-pane>
</body>
```

在*www/js/app.js*中添加*NotifCtrl*如下：

```
.controller('NotifCtrl', ['$scope', '$ionicPlatform','$cordovaLocalNotification', func
tion($scope, $ionicPlatform,$cordovaLocalNotification) {
    $ionicPlatform.ready(function() {
        $scope.notifText = 'Hello World!';
        $scope.triggerNotification = function() {
            $cordovaLocalNotification.schedule({
                id: 1,
                title: 'Dynamic Notification',
                text: $scope.notifText
            }).then(function(result) {
                console.log(result);
            });
        }
    });
}])
```

模拟此Ionic app的时候，会问你是否允许显示通知。允许之后，你可以根据需求分发通知。



> 更多信息参考： http://ngcordova.com/docs/plugins/localNotification/

$cordovaGeolocation

最后一个学习的插件的Geolocation（定位）插件，这个插件可以帮助我们获取设备坐标。
新建一个空白模板项目：

```
ionic start -a "Example 34" -i app.example.thirtyfour example34 blank
```

接下来，给项目添加ngCordova支持。运行如下命令添加定位插件到项目：

```
ionic plugin add cordova-plugin-geolocation
```

启动应用的时候，我们将去获取设备的定位信息。在得到定位信息之前，我们展示一个加载内容信息。 一旦我们得到响应，我们就在页面上显示经度，纬度，以及精确度。
首先，更新*www/index.html*的*body*部分如下：

```
<body ng-app="starter" ng-controller="GeoCtrl">
    <ion-pane>
        <ion-header-bar class="bar-stable">
            <h1 class="title">Ionic Blank Starter</h1>
        </ion-header-bar>
        <ion-content>
            <ul class="list" ng-show="dataReceived">
                <li class="item">
                    Latitude : {{latitude}}
                </li>
                <li class="item">
                    Longitude : {{longitude}}
                </li>
                <li class="item">
                    Accuracy : {{accuracy}}
                </li>
            </ul>
        </ion-content>
    </ion-pane>
</body>
```

之后，在*www/js/app.js*的*run*方法下面添加*GeoCtr*：

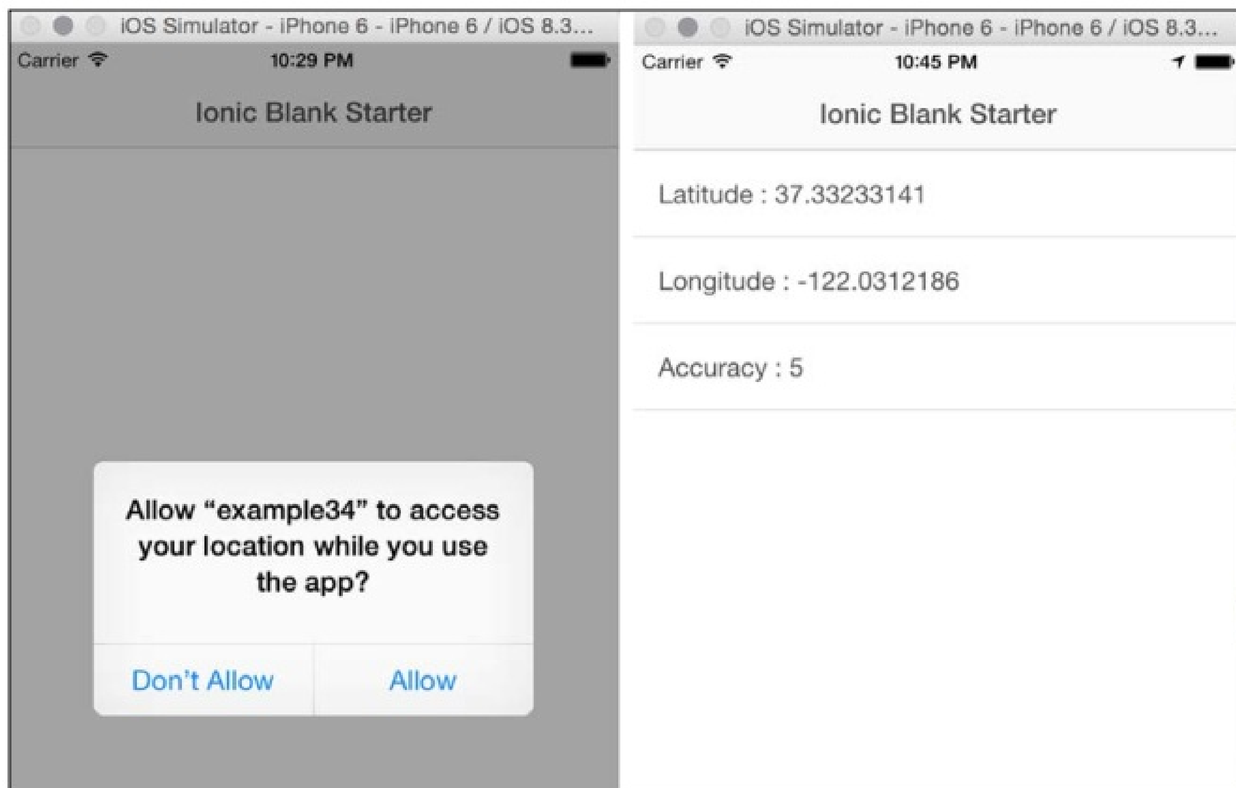
```
.controller('GeoCtrl', ['$scope', '$ionicPlatform','$cordovaGeolocation', '$ionicLoadi
ng', '$timeout', function($scope, $ionicPlatform, $cordovaGeolocation, $ionicLoading,
$timeout) {
    $ionicPlatform.ready(function() {

        $scope.modal = $ionicLoading.show({
            content: 'Fetching Current Location...',
            showBackdrop: false
        });

        var posOptions = {
            timeout: 10000,
            enableHighAccuracy: false
        };

        $cordovaGeolocation
            .getCurrentPosition(posOptions)
            .then(function(position) {
            $scope.latitude = position.coords.latitude;
            $scope.longitude = position.coords.longitude;
            $scope.accuracy = position.coords.accuracy;
            $scope.dataReceived = true;
            $scope.modal.hide();
        }, function(err) {
            // error
            $scope.modal.hide();
            $scope.modal = $ionicLoading.show({
                content: 'Oops!! ' + err,
                showBackdrop: false
            });
            $timeout(function() {
                $scope.modal.hide();
            }, 3000);
        });
    });
}])
```

在模拟此app的时候，你将被询问是否允许访问定位信息。接受之后，就可以看到下面这样的效果：



> 更多信息参考： http://ngcordova.com/docs/plugins/geolocation/

以上范例应该给你很好的演示了如何使用ngCordova。

> 也可以查看其他关于ngCordova其他一些插件的帖子： http://thejackalofjavascript.com/getting-started-withngcordova 完整的插件列表： http://ngcordova.com/docs/plugins/ 当在使用ngCordova的时候，只添加你需要用到的插件。 参考此帖自定义ngCordova： http://ngcordova.com/build/ 记住，在自定义ngCordova之后，就不能用bower下载ngCordova了。

总结

本章中，我们看了什么是Cordova插件，以及他们在Ionic应用中是如何使用的。刚开始，我们为Android和iOS设置了本地开发环境。 然后学习了如何模拟和运行app。接下来，我们探索了如何给Ionic项目添加Cordova插件以及如何使用他们。 最后，在ngCordova的帮助下，我们将插件作为依赖注入我们的Ionic/Angular app中，然后以Angular的方式使用他们。
在下一章中，我们将使用Ionic， ngCordova以及Firebase制作另一个app。
我们将要制作的应用是一个聊天app，用户可以登录其中，查看在线用户。用户可以选中其他用户交换文本，图片以及地理详情等信息。
聊天app的目的是整合Ionic和一个实时数据存储，例如Firebase，同时通过访问设备功能使通讯内容更丰富。

制作一个聊天app

鉴于我们已经学习过了所有移动混合app开发需要的只是，本章我们就真枪实刀的来做一个了。 我们将要制作的是一个信息应用（聊天应用，短信应用），叫做*Ionic Chat*。 第六章 书店*App*中，我们着重整合REST API， 本章我们将要制作的Ionic Chat app更多的关注于利用Ionic整合设备功能，例如摄像头和Geolocation， 同时也会重点关注与实时数据存储（如Firebase）对话。
我们将涵盖如下主题：

- 初步理解Firebase与设置一个Firebase账号
- 了解AngularFire
- 了解应用架构
- 搭建Ionic app并进行编译
- 安装所需插件并整合到Ionic App:8.5.1,8.5.2,8.5.3
- 在设备上测试app

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通: https://github.com/learning-ionic/Chapter-8

Ionic聊天应用

我们本章中将要制作的原因名为Ionic Chat。app的目的是让你熟悉使用AngularFire和Ionic制作的聊天应用，同时使用ngCordova整合Cordova插件与Ionic。
首先我们会学习Firebase，然后聊一点AngularFire，最后学习如何整合AngularFire与Ionic Chat应用。我们将使用Firebase作为实时数据存储。 Firebase将负责实时同步数据。同时我们将使用oAuth Cordova插件与Firebase Auth组合来管理用户的认证。
一旦用户登录后，他将在第一个标签页中看到所有的在线用户。第二个标签页是聊天历史和当前参与聊天的用户组成。 最后，第三个标签页是设置和登出页。
当用户点击聊天列表里的人名的时候，将会打开一个聊天页，聊天页里可以看到过往聊天历史记录，可以发送新的信息，图片以及地理信息给其他用户。

> 为简单起见，应用中我们只展示所有的在线用户。你喜欢的话，可以实现一个“添加好友”功能。

Firebase

Firebase是一个BAAS(Backend As A Service后端即服务)，提供了云后端服务，实时数据存储，用户验证，以及静态主机。

> 更多信息关于Firebase请参考：https://www.firebase.com/features.html

想要快速了解Firebase如何运作，查看以下代码片：

```
var ref = new Firebase("https://<YOUR-FIREBASEAPP>.firebaseio.com");
ref.set({ name: "Arvind Ravulavaru" });
ref.on("value", function(data) {
    var name = data.val().name;
    alert("My name is " + name);
});
```

第一行中，引用了我们的Firebase示例（我们下一部分将会创建）。有了他之后，我们可以设置和保存一个JSON文档到默认的终端。 Firebase作为一个实时数据存储，使用事件驱动的方式管理和同步数据。第3行可以看到，我们订阅了一个value事件，当有新的数据插入到默认终端的时候，这里将会介入。
为更好的理解第3行，想象一下当用户1以及在数据存储中设置了value并注册了value事件。现在，当用户在浏览器中加载这段代码，他先设置value；这样将触发用户1第3行的回调函数以告知用户1用户2的value。
value回调将被调用，传入新加的data值的快照。*data.val*方法将返回新加的值的记录。

> 如果想要代码在页面上正常工作，还需要将Firebase的代码包含进来：

设置Firebase账号

可以通过 https://www.firebase.com/signup/ 申请账号，或者可以使用Github账号登录：https://www.firebase.com/login/

一旦注册或者登录成功之后，你会被带到 https://www.firebase.com/account/#/ ，此处可以添加一个新的项目。输入app名，Firebase会判断名字是否被占用。 例如，当你输入“ionic-chat-app”的时候，你会发现这个名字被占用了（是我为制作本章app而使用的）。

选择一个合适可用的名字点击**Create New App**。这样将会为你创建一个新的app，并附送一个Firebase URL。简单点说这个URL就是你的账户的API KEY。 相对于给用户一堆乱七八糟的字符串作为API密钥来讲，这是非常优雅的解决方案了。

为测试一切是否正常，我们将执行之前的代码片。新建一个文件夹名为*chapter8*,在其中建立另一个文件夹*example35*。然后在*chapter8*中创建一个新文件*index.html*。写入如下代码：

```
<!DOCTYPE html>
<html>
    <head>
    <title>Firebase Test Page</title>
    <script src="https://cdn.firebase.com/js/client/2.2.2/firebase.js"></script>
    </head>
    <body>
        <input type="button" onclick="addNewName()" value="Add New Name">
        <br>
        <ul id="namesList"></ul>
        <script type="text/javascript">
        var ref = new Firebase("https://<YOUR-FIREBASEAPP>.firebaseio.com");
        ref.on('value', function(data) {
            var names = data.val();
            clearList();
            for (var n in names) {
                setName(names[n].name);
            }
        });
        function clearList() {
            document.querySelector('#namesList').innerHTML = '';
        }
        function setName(name) {
            var newName = document.createElement('li');
            newName.innerHTML = 'Name : <b>' + name + '</b>';
            document.querySelector('#namesList').appendChild(newName);
        }
        function addNewName() {
            var name = prompt('Enter Name');
            if (name) {
                // the below statement will save data to the Firebase
                // data store and will invoke the ref.on('value') callback. This will
                // the call the saveName to setdata
                ref.push({
                    'name': name
                });
            }
        }
        </script>
    </body>
</html>
```

在早先的范例中，我们在源文件头部添加了Firebase源文件的引用。在body部分，我们添加了一个按钮，点击此按钮的时候将会弹出提醒，此时用户可以输入他的名字。 用户输入名字完成之后，数据将会以数组的形式保存到Firebase默认的集合中。一旦数据保存完成，*ref.on('value')*事件将会被触发。 一旦触发回调，我们就会清除页面上的HTML元素，重新使用*setName*方法展示名字列表。

你也可以打开一个新的标签页打开相同页面。默认之前添加的值都会展示出来。你这边不用

做任何事情。你可以多添加些数据观察这两个页面的数据同步。
之前范例展示了一个实时数据存储是如何工作的。现在你可以看到Firebase对于我们的聊天应用如何简单易用。

> 当用户输入用户名的时候，我们不直接展示这个值。我们等待他被存放到数据存储中，然后我们等待Firebase调用value事件。在value回调里，我们给用户展示了value。 参考此处理解数据实时更新：https://.firebaseio.com

如果在Firebase上导航到你的app的时候，你可以看到：



所有添加的名字都将直接在你的app名字下展示出来。

AngularFire

鉴于Ionic使用AngularJS作为他的客户端JavaScript框架，我们将使用一个Firebase向的AngularFire以Angular的方式来与Firebase交互。
我们将快速浏览一遍AngularFire，如下：

```
var app = angular.module("nameApp", ["firebase"]);
app.controller("NamesCtrl", function($scope, $firebaseArray) {
    var ref = new Firebase("https://<YOUR-FIREBASEAPP>.firebaseio.com/names");
    // create a synchronized array
    $scope.names = $firebaseArray(ref);

    $scope.addName = function() {
        $scope.names.$add({
            text: $scope.newName
        });
        };
});
```

首先我们将新建一个AngularJS应用，然后添加了firebase作为依赖。然后我们创建了一个控制器，注入*$firebaseArray*作为依赖。 一旦调用控制器的时候，我们将创建一个Firebase App的引用。此时，我们将使用name创建一个子集或者内置集然后保存，而不是将他直接存放到根集合。
将*$firebaseArray(ref)* 指派给 *$scope.names* 就可以将他变为同步集合了。简单来说，如果数据存储里面的数据变更的时候，我们的scope变量将会自动同步更新，同时触发视图或者模板里面的更新。 这种现象也称为三方数据（Three-Way Data）绑定。

> 更多关于三方数据绑定，请参考： https://www.firebase.com/blog/2013-10-04-firebase-angular-databinding.html 如果想要上面的代码正常执行，需要在你的页面上导入Firebase，AngularJS，以及AngularFire脚本文件。

我们将实现快速实现一个范例来展示AngularFire是如何工作的。新建一个文件夹名为*example36*，在其中新建一个文件名为*index.html*，在其中加入代码如下：

```
<!DOCTYPE html>
<html>
<head>
    <title>AngularFire Test Page</title>
    <script src="https://cdn.firebase.com/js/client/2.2.2/firebase.js">
    </script>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js
"></script>
    <script src="https://cdn.firebase.com/libs/angularfire/1.1.1/angularfire.min.js"><
/script>
</head>
    <body ng-app="NamesApp" ng-controller="NamesCtrl">
        <input type="button" ng-click="addNewName()" value="Add New Name">
        <br>
        <ul>
            <li ng-repeat="n in names">
            Name : <b> {{n.name}} </b>
            </li>
        </ul>
        <script type="text/javascript">
        var app = angular.module("NamesApp", ["firebase"]);
        app.controller("NamesCtrl", function($scope, $firebaseArray) {
            var ref = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com/names")
;

            // create a synchronized array
            $scope.names = $firebaseArray(ref);

            $scope.addNewName = function() {
                var name = prompt('Enter Name');
                if (name) {
                    $scope.names.$add({
                        name: name
                    });
                };
            }
        });
        </script>
    </body>
</html>
```

以上范例中，我们引入了Firebase，AngularJS以及AngularFire的源文件。

> 一定要记住在Firebase和AngularJS之后加载AngularFire。

我们新建了一个模组名为*NamesApp*，添加了一个新的控制器名为*NamesCtrl*。我们的HTML由一个重复scope里面的names数组的ng-repeat组成。names变量是一个同步数组。
当用户点击**Add New Name**按钮的时候，我们给用户提醒他在何处输入一个名字。一旦名字输入完成，我们将使用我们的名字同步数组的*$add*方法将新对象推送到数据存储。 然后

Firebase负责在数据之间进行同步。
现在，当你打开 *https://.firebaseio.com* 你将看到：

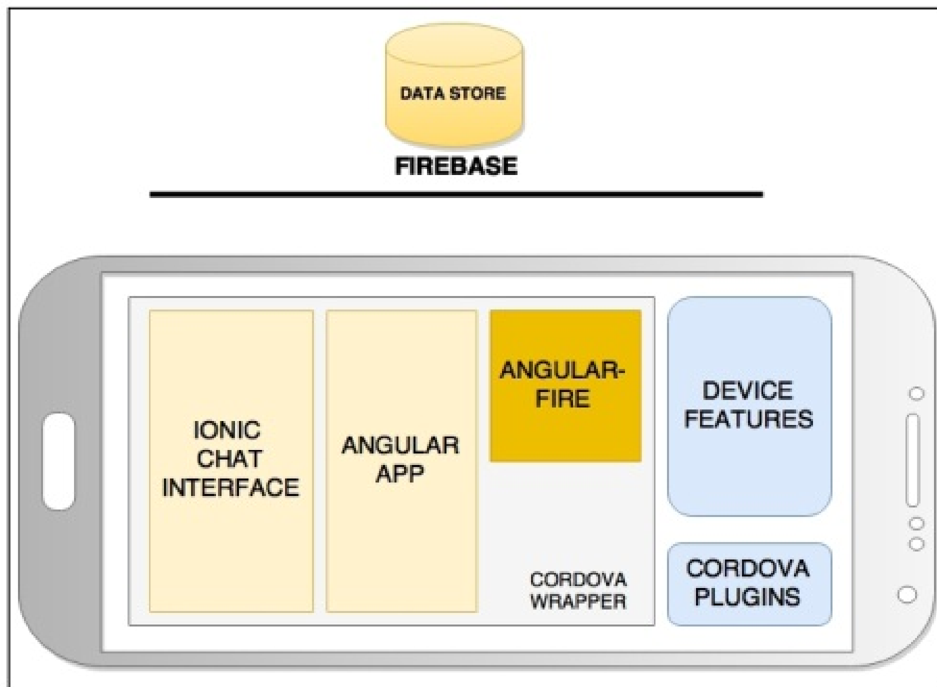


这次，数据将被添加到一个子对象或者内置对象的names属性上了。

> 在运行范例之前，我已经删掉了旧数据。如果想利用Firebase制作一个**Create**，**Read**，**Update**和**Delete**（CRUD，增删改查）的应用的话，参考：http://thejackalofjavascript.com/getting-started-withfirebase/

应用架构

现在我们熟悉了Firebase和AngularFire，我们就来看一下app如何设计：



如上图所示，我们将使用Firebase作为数据存储。在Ionic应用中使用AngularFire来与Firebase交互。Ionic应用同时也使用ngCordova与Cordova插件交互以使用设备功能。在我们将要制作的聊天应用中，Firebase的角色是管理聊天数据。在我们的聊天应用中，我们需要在Firebase集合中创建两个终端：

- 在线用户：这个终端存储所有的在线用户
- 聊天：这个终端将存储两个用户之前的聊天信息

作为聊天应用的一部分，我们允许用户以下操作：

- 发送文本信息
- 从相册分享图片
- 照相并分享
- 分享用户地理位置 我们将把所有数据都存放到Firebase中。你是不是在想，图片要哪里存咧？好吧，这就是聪明之处了；我们将会把图片转成base64格式然后将*base64-encode*的字符串存放到Firebase。 至于用户位置信息，我们只保存用户的坐标，然后在其他用户界面上复制坐标就可以了。

  > 也可以使用Google Static Maps API（谷歌静态地图API）来分享地理信息：https://developers.google.com/maps/documentation/staticmaps/

认证

我们将使用Google Open ID认证服务来认证用户。同时我们也会使用Cordova的*ng-cordova-oauth*插件与Firebase的oAuth组合来做认证服务。 Cordova插件通过浏览器内置插件来管理弹出认证和获取token。然后此token会被传到Firebase认证以及建立一个session。这些在我们开始开发app的时候会越来越清晰。

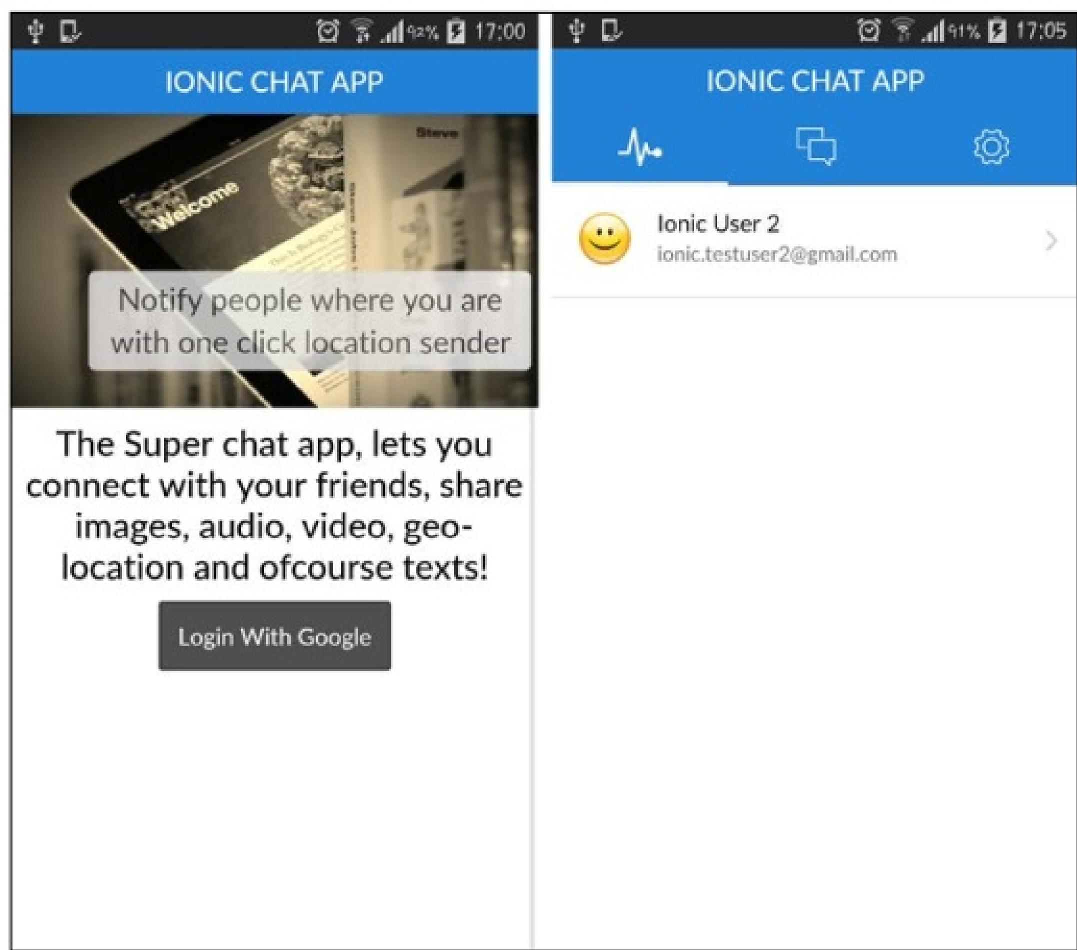
应用工作流

用户启动app的时候，我们将展示应用首页，有一个滑动框和一个**Login**按钮。

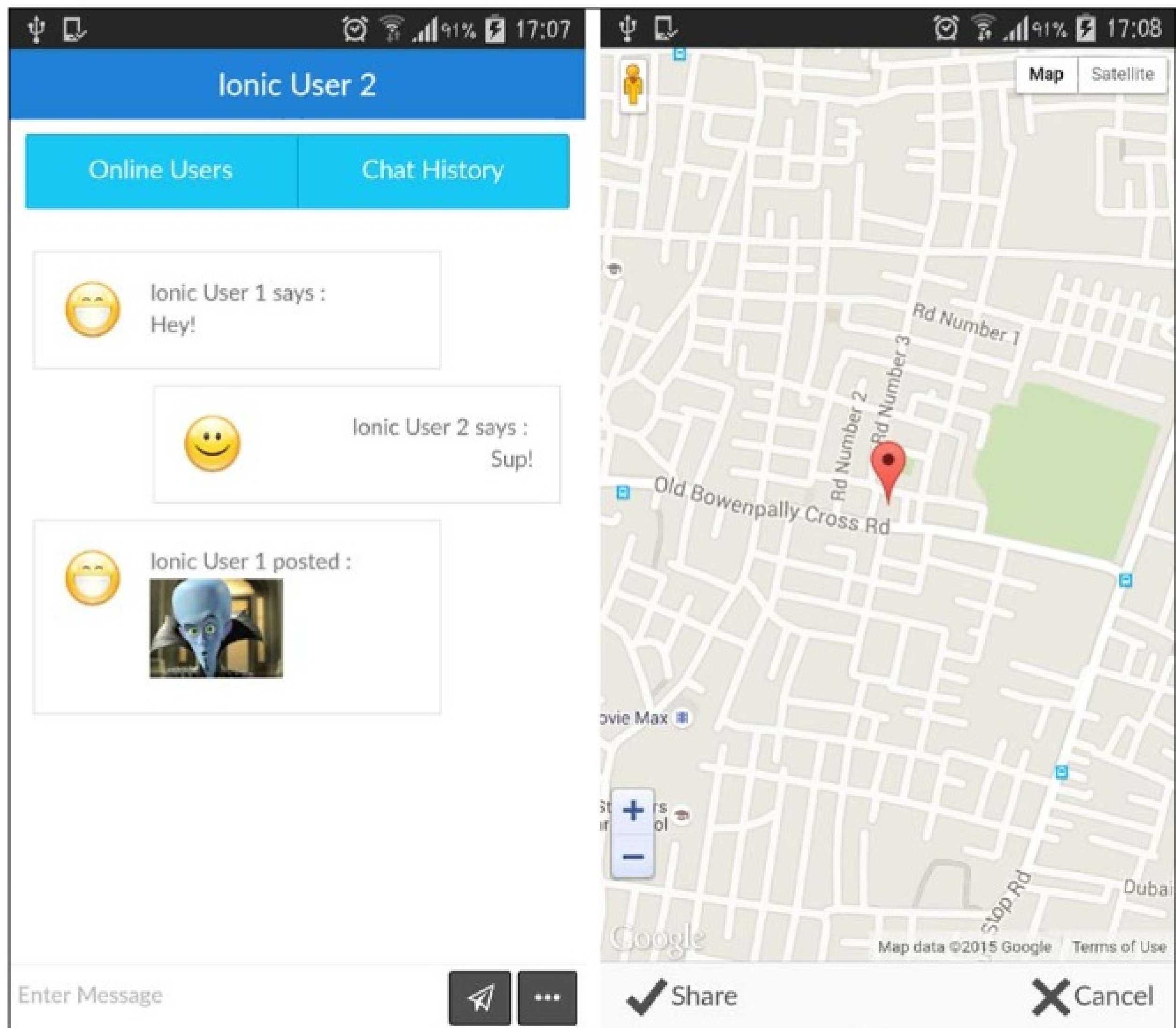
到目前为止应用只支持Google登录。用户点击**Login**按钮的时候，他将被重定向到Google登录页面。登录成功之后，app提供了访问之后，用户将被定向到首页，在这里Google oAuth提供的token将被发送到Firebase来建立session（会话）。 然后，用户将被重定向到有三个标签页的页面。
标签页一是由所有在线用户组成。标签页二是由当前正在聊天中的用户组成，最后，标签页三是由设置屏幕和**Lougout**选项所组成。
当用户点击标签页一或者标签页二的列表中的用户的时候，用户将被带到聊天页面，此页将展示与选中用户之间的聊天记录（如果有的话）。
这样设计app以使事情简单化并能涵盖其他一些主题，可以帮助你更好的理解移动混合应用开发的完整生态系统。

app预览

继续进行之前，我们先快速查看一下最终输出，因为在开发完成之前我们是看不到任何输出结果的。
下图左边展示的是登录 **Login**屏幕。下图右边显示的一个三个标签页的是主页 **Home**屏幕：


17:00
IONIC CHAT APP
Notify people where you are with one click location sender
The Super chat app, lets you connect with your friends, share images, audio, video, geo-location and ofcourse texts!
Login With Google
17:05
IONIC CHAT APP
Ionic User 2
ionic.testuser2@gmail.com

下图左边展示的是聊天界面，右边显示的地图屏幕是用户分享的地理位置信息的展示：



数据结构

默认每次集合增加新的数据集的时候，Firebase都会通知所有的客户端。当我们创建一个普通的聊天时的时候，我们只需要通知所有连接到app的用户，不用做其他多余的事情。
但是，由于我们是关注一对一的聊天应用，我们需要一些逻辑来帮助我们管理参与双方的用户之间的通讯。
当用户登录的时候，我们使用当前用户详情来更新在线用户集合。这将会通知所有的已登录用户。一旦某用户离线的时候，我们从在线用户集合中移除此用户。

> 我们使用Firebase Authentication来存储登录信息。在Forge中看不到任何已注册用户信息。 现在用户已登录，那么他将出现在在线用户中。如果用户B想要和用户A聊天，我们需要创建一个新的终端，仅供用户A和用户B通讯。

创建一个新的动态终端的逻辑有点复杂，步骤如下：

1. 分辨用户A和用户B的邮件地址

2. 对用户A和用户的邮件地址执行哈希函数。当我们以不同顺序传入相同数组的时候，这个哈希函数都会返回同一个字符串
3. 使用以上哈希字符串，我们在聊天集合内创建一个新的终端
4. 如果用户B对用户A发起了聊天，用户B将创建一个动态终端，根据用户B向用户A发送的第一个信息，将会触发在聊天集合内运行一个监听器。 这个监听器将会检查信息发送对象是否是已登录用户。如果是，将会通知用户A。

有点复杂，但是却能很好的工作。

> 我在一个node-webkit桌面版聊天应用中实现了相同的逻辑。可以在此处查看：http://thejackalofjavascript.com/one-to-one-chat-client/ 现在我们对数据如何架构有了想法，我们需要考虑一下要用到哪些Cordova插件。

Cordova 插件

我们将要用到以下插件（除了模块自动下载的之外）：

- cordova-plugin-inappbrowser：这个用来管理Google认证
- cordova-plugin-media-capture：用来照相并与其他用户分享
- com.synconset.imagepicker：用来从相册获取一张图片
- cordova-plugin-file：在将图片转成base64的时候与文件系统交互
- cordova-plugin-geolocation：用来获取用户的Geo坐标 插件会在具体使用的时候学习。

Github上的代码

本章代码已经开源在Github上：https://github.com/learning-ionic 如果有任何问题可以发起issue，我们尽力为大家解答。我也会解决读者报告的所有bug。

开发应用

首先，我们将新建和设置app。

新建和设置app

运行如下命令，新建一个标签页应用：

```
ionic start -a "Ionic Chat App" -i app.ionic.chat ionic-chat-app tabs
```

通过*cd*命令，进入*ionic-chat-app*文件夹运行：

```
ionic serve
```

然后就可以看到标签页范例的app了。
继续深入之前，我们将通过Bower安装应用所需的依赖。在项目的根目录下，运行：

```
bower install ngCordova ng-cordova-oauth firebase angularfire lato --save
```

这些bower组件的使用主旨：

- ngCordova：ngCordova库
- ng-cordova-oauth：编写本书的时候，*ng-cordova-oauth*有个问题是与ngCordova捆绑的问题，所以我们另外安装和使用他。 我现在面对的这个问题在你学习本书的时候可能已经解决了。
- firebase：这是firebase的源代码
- angularfire：这是AngularFire的源代码
- lato：Lato字体（(https://www.google.com/fonts/specimen/Lato）

> 我没从Google加载Lato字体，因为我已经在本机安装了。这样就确保了在本机没有联网的情况下字体可以正常使用。 你也可以参考*localFont*来实现几秒钟内本地存储Web字体的缓存（https://github.com/jaicab/localFont），这样也可以在不用本机安装的情况下正常显示字体。

接下来，我们将给项目添加SCSS支持；运行如下：

```
ionic setup sass
```

现在，我们给下载下来的依赖库添加引用。在*index.html*中进行如下变更。
首先，我们先看一下*ng-cordova*和*ng-cordova-oauth*。以下两个*script*标签是在Ionic bundle之后，*cordova.js*之前：

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
<script src="lib/ng-cordova-oauth/dist/ng-cordovaoauth.js"></script>
```

我们也要在app添加指令来管理地图。我们稍后会添加这个指令，但是现在只要添加引用就可以了。
在*services.js*文件的引用之后添加如下*script*标签：

```
<script src="js/directives.js"></script>
```

接下来，我们需要添加Google地图API的引用。在*head*标签的最后添加如下脚本：

```
<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDgE3k3per7mf0qjZLWwlbMX
QL1OhH-x44&sensor=true"></script>
```

> 我将给你展示如何在使用Google认证设置里面获取你自己的Google API key（上面脚本中的script标签）

最后，我们将添加Lato字体。在*ionic.app.css*的引用之上，添加：

```
<link href="lib/lato/css/lato.min.css" rel="stylesheet">
```

> 时过境迁，上面参考的资源可能已经不是如今的路径了。如果资源出现“not found（404）”错误，在*lib*文件夹里面重新检查他的位置。

我们会将*body*标签上的模组名从*starter*改为*IonicChatApp*。 接着，将*nav bar*类型从*bar-stable*改为*bar-positive*。
做完这些，我们就完成了*index.html*的设置。
接下来，打开*www/js/app.js*。由于我们在index页面上对模组进行了重命名操作，我们在*app.js*也需要对他进行重命名。修改后的AngularJS模组声明如下：

```
angular.module('IonicChatApp', ['ionic', 'chatapp.controllers','chatapp.services', 'ch
atapp.directives', 'ngCordova','ngCordovaOauth', 'firebase'])
```

我们也重命名了控制器和服务的命名空间，在指令模组，*ngCordova*, *ngCordovaOayth*和*Firebase*中添加引用。

> 注意，我们将*ngCordovaOauth*模组作为依赖添加到了主模组。这是因为打包版（ng-cordova.js）在本章编写的时候有个issue。 如果你使用打包版的Cordova oAuth插件；你就不需要包含此依赖库和他的源代码了。

> 用到的名词：
>
> - authentication 授权，认证

安装所需的Cordova插件

我们现在安装所需插件，运行如下命令：

```
ionic plugin add https://github.com/wymsee/cordova-imagePicker.git

ionic plugin add cordova-plugin-file

ionic plugin add cordova-plugin-geolocation

ionic plugin add cordova-plugin-inappbrowser

ionic plugin add cordova-plugin-media-capture
```

获取Google API key

由于我们的app用来Google OAuth，我们需要一个Client ID。可通过以下步骤获取一个Client ID:

1. 导航至：https://console.developers.google.com
2. 点击 **Create project** ，输入项目名
3. 创建完成后打开项目
4. 在左边的菜单中点击 **APIs and auth** 然后点击 **Consent** 。填写必需信息。产品名是强制的。
5. 点击左边的菜单中的 **APIs and auth**，然后点击 **Credentials**
6. 点击OAuth部分下面的**Create new Client ID**
7. 选中如下：
    - 应用类型为网页应用（web application）
    - 授权（Authorized）JavaScript origins为 http://localhost
    - 授权（Authorized）重定向URI为 http://localhost/callback
8. 一旦上述信息填写完成，点击**Creat Client ID**你就可以看到网页应用的Client ID

有了Client ID之后，你就需要在我们的Firebase Forge中更新。导航到Firebase应用页（查看实时数据更新的地方）。在页面左边，可以找到一个菜单选项“**Login and Auth**；点击之。 当右边页面刷新之后，点击Google标签页，填入之前生成的Google Client ID和Secret。

> 以上步骤对认证工作非常重要

如果你想创建一个API key来访问Google地图API，可以使用以下步骤：

1. 点击左边菜单的**APIs and auth**，然后点击**Credentials**
2. 点击**Public API Access**下面的**Create new Key**
3. 从弹出框里面选择浏览key
4. 将**Accept request from these HTPP referrers**文本域留空
5. 点击**Create**。这样将生成API key，这一用这个API key来替换*index.html*中的那个key
6. 点击左边菜单中的**APIs and auth**，然后点击**APIs**
7. 在搜索框里面搜索**Google Maps JavaScript API v3**，点击搜索到的连接
8. 点击**Enable API**按钮激活API

现在，我们有了Client ID，我们将设置一些常量。 在*www/js/app.js***中，run方法之后config* 方法之前，添加以下3个常量：

```
.constant('FBURL', 'https://ionic-chat-app.firebaseio.com/')
.constant('GOOGLEKEY', '1002599169952-4uchnlc7ahm6ng4696p9tgr1adhsiqv5.apps.googleuser
content.com')
.constant('GOOGLEAUTHSCOPE', ['email'])
```

将FBURL替换为你的Firebase应用URL。将Google key替换为我们早先生成的Client ID。作为OAuth请求的一部分，我们需要向Google发送一个scope， 这样我们就可以取回我们需要的信息。我们的应用只需要用户的基本信息；此外，我们用邮件作为一个scope。

设置路由以及路由认证

由于我们建立的是一个标签页模板的应用，因为我们就基本有了所有所需的路由。我们将对已有的路由做一些变更并给他们添加认证。 这样，当授权失败的时候，视图就不会展示。我们将使用路由的*resolve*属性来实现。

> 继续深入之前，我建议先浏览一下AngularJS状态路由器的*resolve*属性。这个属性负责在加载控制器之前解析所有指定的promise。 这个在我们加载控制器之间验证用户授权状态非常有用。更多信息参考： https://github.com/angularui/ui-router/wiki#resolve

Firebase的*$firebaseAuth*函数上来两个方法：

- *$waitForAuth*：他返回一个将用做解析当前授权状态的promise。这个仅用在主页路由上。
- *$requireAuth*：他返回一个与当前授权状态一切解析的promise；否则，将拒绝这个promise。这个将用在每个需要授权认证的路由。

我们将平衡借助这两个方法来控制未授权和授权的用户可以看到什么不可以看到什么。 我们将为现有的路由多加一个路由名为*main*。这个路由将作为默认路由且作为我们应用的主页。 我们在每个路由上添加一个*resolve*属性用来解析Firebase Auth。 同时，修改chat-detail

路由使他摆脱chats路由的子路由的身份。
更新后的路由部分代码如下：

```
$stateProvider.state('main', {
    url: '/',
    templateUrl: 'templates/main.html',
    controller: 'MainCtrl',
    cache: false,
    resolve: {
    'currentAuth': ['FBFactory', 'Loader',  function(FBFactory, Loader) {
        Loader.show('Checking Auth..');
        return FBFactory.auth().$waitForAuth();
    }]
    }
})

.state('tab', {
    url: "/tab",
    abstract: true,
    cache: false,
    templateUrl: "templates/tabs.html"
})
.state('tab.dash', {
    url: '/dash',
    cache: false,
    views: {
        'tab-dash': {
            templateUrl: 'templates/tab-dash.html',
            controller: 'DashCtrl'
        }
    },
    resolve: {
        'currentAuth': ['FBFactory', function(FBFactory) {
            return FBFactory.auth().$requireAuth();
        }]
    }
})

.state('tab.chats', {
    url: '/chats',
    cache: false,
    views: {
        'tab-chats': {
            templateUrl: 'templates/tab-chats.html',
            controller: 'ChatsCtrl'
        }
    },
    resolve: {
        'currentAuth': ['FBFactory', function(FBFactory) {
            return FBFactory.auth().$requireAuth();
        }]
    }
```

```
})

.state('tab.account', {
    url: '/account',
    cache: false,
    views: {
        'tab-account': {
            templateUrl: 'templates/tab-account.html',
            controller: 'AccountCtrl'
        }
    },
    resolve: {
        'currentAuth': ['FBFactory', function(FBFactory) {
            return FBFactory.auth().$requireAuth();
        }]
    }
})

.state('chat-detail', {
    url: '/chats/:otherUser',
    templateUrl: 'templates/chat-detail.html',
    controller: 'ChatDetailCtrl',
    cache: false,
    resolve: {
        'currentAuth': ['FBFactory', 'Loader',
        function(FBFactory, Loader) {
            Loader.show('Checking Auth..');
            return FBFactory.auth().$requireAuth();
        }]
    }
});

$urlRouterProvider.otherwise('/');
```

在如上代码中使用工厂的时候我们会设置*FBFactory*和*Loader*。

> 注意，我们将otherwise的路由更新为'/'

此时，当*$requireAuth*拒绝promise的时候，以为这用户去到了一个需要认证的页面，但是用户并没有认证。 因为，我们需要在这个时候添加一个监听器，将用户重定向到登录页。当Firebase Auth拒绝了promise的时候，他发出了一个*stateChangeError*事件。 我们将在*run*方法中监听这个事件，然后将用户重定向到主页。
在*$ionicPlatform.ready*方法中，添加以下*stateChangeError*事件到*run*方法：

```
$rootScope.$on('$stateChangeError', function(event, toState,toParams, fromState, fromP
arams, error) {
    if (error === 'AUTH_REQUIRED') {
        $state.go('main');
    }
});
```

> 记住给*run*方法注入*$rootScope*和*$state*依赖。

接下来，我们将使用*$ionicConfigProvider*来设置一些默认值。
给*config*方法添加*$ionicConfigProvider*作为依赖。在开始初始化路由之前，将以下代码片段添加到*config*方法内：

```
$ionicConfigProvider.backButton.previousTitleText(false);
$ionicConfigProvider.views.transition('platform');
$ionicConfigProvider.navBar.alignTitle('center');
```

> 上面的配置也不是必需的。只是给你展示如何在一个实时app中使用*$ionicConfigProvider*

设置服务/工厂

现在我们设置好了主体应用，我们现在就来看看需要用到的工厂了。工厂都将添加到*www/js/services.js*中。可以先打开这个文件，清空其中所有内容。
首先，添加*chatapp.services*模组和一个与*localStorage*交互的工厂：

```
angular.module('chatapp.services', [])
    .factory('LocalStorage', [function() {
        return {
            set: function(key, value) {
                return localStorage.setItem(key, JSON.stringify(value));
            },

            get: function(key) {
                return JSON.parse(localStorage.getItem(key));
            },

            remove: function(key) {
                return localStorage.removeItem(key);
            },
        };
    }])
```

接下来，添加一个工厂用来管理Ionic加载服务：

```
.factory('Loader', ['$ionicLoading', '$timeout',function($ionicLoading, $timeout) {
        return {
            show: function(text) {
                //console.log('show', text);
                $ionicLoading.show({
                    content: (text || 'Loading...'),
                    noBackdrop: true
                });
            },

            hide: function() {
                //console.log('hide');
                $ionicLoading.hide();
            },
            toggle: function(text, timeout) {
                var that = this;
                that.show(text);
                $timeout(function() {
                    that.hide();
                }, timeout || 3000);
            }
        };
    }
])
```

也要创建一个与Firebase交互的工厂：

```
.factory('FBFactory', ['$firebaseAuth', '$firebaseArray', 'FBURL','Utils',function($fi
rebaseAuth, $firebaseArray, FBURL, Utils) {
        return {
        auth: function() {
            var FBRef = new Firebase(FBURL);
            return $firebaseAuth(FBRef);
        },

        olUsers: function() {
            var olUsersRef = new Firebase(FBURL + 'onlineUsers');
            return $firebaseArray(olUsersRef);
        },

        chatBase: function() {
            var chatRef = new Firebase(FBURL + 'chats');
            return $firebaseArray(chatRef);
        },

        chatRef: function(loggedInUser, OtherUser) {
            var chatRef = new Firebase(FBURL + 'chats/chat_' + Utils.getHash(OtherUser
, loggedInUser));
            return $firebaseArray(chatRef);
        }
        };
    }
])
```

上面的代码中，*olUsersRef*将Firebase的引用指向 https://ionicchat-app.firebaseio.com/onlineUsers ， *chatBase*指向 https://ionic-chatapp.firebaseio.com/chats ， *chatRef*指向在两个用户之间动态创建的终端。

我们也会创建一个用户工厂用来存储用户信息，在线用户和已有ID。已有ID是在 https://ionicchat-app.firebaseio.com/onlineUsers中创建的入口object ID。 这个已有ID将在用户离线时，*onlineUsers*集合删除对象之用：

```
.factory('UserFactory', ['LocalStorage', function(LocalStorage) {
    var userKey = 'user',
    presenceKey = 'presence',
    olUsersKey = 'onlineusers';

    return {
        onlineUsers: {},

        setUser: function(user) {
            return LocalStorage.set(userKey, user);
        },

        getUser: function() {
            return LocalStorage.get(userKey);
        },
```

```
        cleanUser: function() {
            return LocalStorage.remove(userKey);
        },

        setOLUsers: function(users) {
            // >> we need to store users as pure object.
            // else we lose the $ method of FB.
            // >> sometime, onlineUsers becomes null while
            // navigating between tabs, so we save a copy in LS
            LocalStorage.set(olUsersKey, users);
            return this.onlineUsers = users;
        },

        getOLUsers: function() {
            if (this.onlineUsers && this.onlineUsers.length > 0) {
                return this.onlineUsers
            } else {
                return LocalStorage.get(olUsersKey);
            }
        },

        cleanOLUsers: function() {
            LocalStorage.remove(olUsersKey);
            return onlineUsers = null;
        },

        setPresenceId: function(presenceId) {
            return LocalStorage.set(presenceKey, presenceId);
        },

        getPresenceId: function() {
            return LocalStorage.get(presenceKey);
        },

        cleanPresenceId: function() {
            return LocalStorage.remove(presenceKey);
        },
    };
}])
```

最后，添加一些工具方法：

```
.factory('Utils', [function() {
    return {

        escapeEmailAddress: function(email) {
            if (!email) return false
            // Replace '.' (not allowed in a Firebase key) with ','
            email = email.toLowerCase();
            email = email.replace(/\./g, ',');
            return email.trim();
        },

        unescapeEmailAddress: function(email) {
            if (!email) return false
            email = email.toLowerCase();
            email = email.replace(/,/g, '.');
            return email.trim();
        },

        getHash: function(chatToUser, loggedInUser) {
            var hash = '';
            if (chatToUser > loggedInUser) {
                hash = this.escapeEmailAddress(chatToUser) + '_' + this.escapeEmailAdd
ress(loggedInUser);
            } else {
                hash = this.escapeEmailAddress(loggedInUser) + '_'  + this.escapeEmail
Address(chatToUser);
            }
            return hash;
        },

        getBase64ImageFromInput: function(input, callback) {
            window.resolveLocalFileSystemURL(input, function(fileEntry) {
                fileEntry.file(function(file) {
                    var reader = new FileReader();
                    reader.onloadend = function(evt) {
                        callback(null, evt.target.result);
                    };
                    reader.readAsDataURL(file);
                },
                function() {
                    callback('failed', null);
                });
            },
            function() {
                callback('failed', null);
            });
        }
    };
}])
```

*getHash*方法接收两个邮件地址返回一个哈希。这个方法用作构建动态终端。
*getBase64ImageFromInput*用作将图片转换成base64编码的字符串，以存放到Firebase。
做完这些之后，我们完成了工厂的设置。

设置地图指令

由于我们需要给用户提供当前位置分享的功能呢，我们需要一个地图指令来将坐标以一个可呈现的方式显示出来。 我们从地图模块（https://github.com/driftyco/ionicstarter-maps/blob/master/js/directives.js）中借用*map*指令，按需修改。
在*www/js*文件夹内新建一个文件名为*directives.js*。更新其内容如下：

```
angular.module('chatapp.directives', [])
  .directive('map', function() {
    return {
        restrict: 'E',
        scope: {
            onCreate: '&'
        },

        link: function($scope, $element, $attr) {
            function initialize() {
                var lat = $attr.lat || 43.07493;
                var lon = $attr.lon || -89.381388;
                var myLatlng = new google.maps.LatLng(lat, lon);

                var mapOptions = {
                    center: myLatlng,
                    zoom: 16,
                    mapTypeId: google.maps.MapTypeId.ROADMAP
                };

                if ($attr.inline) {
                    mapOptions.disableDefaultUI = true;
                    mapOptions.disableDoubleClickZoom = true;
                    mapOptions.draggable = true;
                    mapOptions.mapMaker = true;
                    mapOptions.mapTypeControl = false;
                    mapOptions.panControl = false;
                    mapOptions.rotateControl = false;
                }

                var map = new google.maps.Map($element[0],    mapOptions);
                // custom function to manage markers
                map.__setMarker = function(map, lat, lon) {
                    var marker = new google.maps.Marker({
                        map: map,
                        position: new google.maps.LatLng(lat, lon)
                    });
```

```
                }

                $scope.onCreate({
                    map: map
                });

                map.__setMarker(map, lat, lon);
            }
            if (document.readyState === 'complete') {
                initialize();
            } else {
                google.maps.event.addDomListener(window, 'load',   initialize);
            }
        }
    }
});
```

我所做的修改是：调整map指令以在聊天信息中的内联地图上使用内联属性能够正常运行。我也添加了展示标记的支持。

设置控制器

先来，设置每个路由的控制器。打开*www/js/controller.js*清除其中所有内容。添加一个新模组名为*chatapp.controllers*：

```
angular.module('chatapp.controllers', [])
```

给*chatapp.controllers*模组添加一个*run*方法。*run*方法由监视进入的聊天信息的逻辑所组成。我们利用聊天基本URL建立了一个连接，然后监听了他的任何改变。 如果有新的聊天添加或者聊天内容变更，我们将辨别新的聊天信息是否跟当前用户相关。如果是的话，我们广播一个*newChatHistory*事件，这样在聊天历史标签页中，我们将会展示新的聊天信息：

```
.run(['FBFactory', '$rootScope', 'UserFactory', 'Utils',function(FBFactory, $rootScope
, UserFactory, Utils) {
    $rootScope.chatHistory = [];

    var baseChatMonitor = FBFactory.chatBase();
    var unwatch = baseChatMonitor.$watch(function(snapshot) {
        var user = UserFactory.getUser();
        if (!user) return;
        if (snapshot.event == 'child_added' || snapshot.event == 'child_changed') {
            var key = snapshot.key;
            if (key.indexOf(Utils.escapeEmailAddress(user.email)) >= 0) {
                var otherUser = snapshot.key.replace(/_/g,'').replace('chat', '').repl
ace(Utils.escapeEmailAddress(user.email),  '');

                if ($rootScope.chatHistory.join('_').indexOf(otherUser) === -1) {
                    $rootScope.chatHistory.push(otherUser);
                }

                $rootScope.$broadcast('newChatHistory');
                /*
                * TODO: PRACTICE
                * Fire a local notification when a new chat
                comes in.
                */
            }
        }
    });
}
])
```

> 为对以上代码更好的理解，我没有实现本地通知那部分。如果你愿意的接受的话，你的任务就是在聊天送到当前用户的时候实现本地通知。通知内容是由聊天信息和来源用户所组成。 当点击通知的时候，将会把当前用户带到聊天界面。

接下来，我们将开始*MainCtrl*。*MainCtrl*与我们一样的主视图连接。将以下的*MainCtrl*定义添加到*www/js/controllers.js*文件里的*run*方法后面：

```
.controller('MainCtrl', ['$scope', 'Loader', '$ionicPlatform','$cordovaOauth', 'FBFact
ory', 'GOOGLEKEY', 'GOOGLEAUTHSCOPE','UserFactory', 'currentAuth', '$state',
function($scope, Loader, $ionicPlatform, $cordovaOauth,FBFactory, GOOGLEKEY, GOOGLEAUT
HSCOPE, UserFactory, currentAuth,$state) {
    $ionicPlatform.ready(function() {
        Loader.hide();
        $scope.$on('showChatInterface', function($event,authData) {
            if (authData.google) {
                authData = authData.google;
            }

            UserFactory.setUser(authData);
            Loader.toggle('Redirecting..');
            $scope.onlineusers = FBFactory.olUsers();

            $scope.onlineusers.$loaded().then(function() {
                $scope
                .onlineusers
                .$add({
                    picture:authData.cachedUserProfile.picture,
                    name: authData.displayName,
                    email: authData.email,
                    login: Date.now()
                })
                .then(function(ref) {
                    UserFactory.setPresenceId(ref.key());
                    UserFactory.setOLUsers($scope.onlineusers);
                    $state.go('tab.dash');
                });
            });
            return;
        });

        if (currentAuth) {
            $scope.$broadcast('showChatInterface',currentAuth.google);
        }

        $scope.loginWithGoogle = function() {
            Loader.show('Authenticating..');
            $cordovaOauth.google(GOOGLEKEY, GOOGLEAUTHSCOPE).
             then(function(result) {
                FBFactory.auth()
                  .$authWithOAuthToken('google',result.access_token)
                  .then(function(authData) {
                    $scope.$broadcast('showChatInterface',authData);
                }, function(error) {
                    Loader.toggle(error);
                });
            }, function(error) {
```

```
                Loader.toggle(error);
            });
        }
    });
}
])
```

当promise在路由的resolve方法中解析完成的时候，*MainCtrl*将会被调用，并给他传入*currentAuth*作为依赖。如果用户已经登录的话，*currentAuth*将等于*oauth*；相反，则为*null*。
我们在*$scope*上注册了*showChatInterface*。这个方法在用户已经登录（具体来说就是*currentAuth*不为*null*）或者用户明确登录的时候被调用。当此事件触发的时候，我们通过*UserFactory.setUser*方法将用户数据保存到*localStorage*。 一旦这个操作完成了，我们将发起一个请求以获取所有的在线用户。拿到在线用户列表之前，我们就将当前用户详情添加到*onlineUsers*集合中。添加完成之后，我们在*localStorage*中*setPresenceiD*和*setOLUsers*然后将用户重定向到聊天界面。

当用户的点击**Login with Google**按钮的时候，*$scope.loginWithGoogle*方法将被触发。我已经添加了Firebase Auth和*cordovaOauth*插件以展示如何同时使用他们。 如果你觉得复杂的话，你可以直接使用Firebase Auth登录用户，而不是从*cordovaOauth*获取token然后使用Firebase *$authWithOAuthToken*认证用户。
一旦认证成功，我们将广播一个*showChatInterface*事件，这个事件将会保存数据并重定向用户。

> 作为范例的一部分，我只实现了Google OAuth*。作为练习，你可以整合其他oAuth提供者。

一旦用户登录成功，他将被重定向到标签页界面。默认的标签页是dashboard标签页，他与*DashCtrl*连接。
*DashCtrl*的目标是取得在线用户列表并展示。他需要这样子的代码：

```
.controller('DashCtrl', ['$scope', 'UserFactory','$ionicPlatform', '$state', '$ionicHi
story',function($scope, UserFactory, $ionicPlatform, $state,$ionicHistory) {
    $ionicPlatform.ready(function() {
        $ionicHistory.clearHistory();

        $scope.users = UserFactory.getOLUsers();
        $scope.currUser = UserFactory.getUser();
        var presenceId = UserFactory.getPresenceId();

        $scope.redir = function(user) {
            $state.go('chat-detail', {
                otherUser: user
            });
        }
    });
}
])
```

> 在上面的控制器代码中我使用了*$ionicHistory.clearHistory*方法。这样就保证了，当用户登录成功并且在标签页上的时候，点击设备的返回按钮不会将用户带回登录界面而是直接退出app。 因此，使用*$ionicHistory.clearHistory*方法我们清空历史记录，然后设备的返回按钮将关闭应用。

接下来我们将使用中间位置的标签页，这个标签页用来展示聊天记录的。由于这只是一个范例应用，所以我们不需要严格的维护任何用户资料。我们直接从auth对象里面获取值，然后建立UI。 我们将使用相同的逻辑在历史标签页中显示用户信息。用户对象是从在线用户列表中获取的。

*ChatsCtrl*代码如下，添加到*DashCtrl*后面：

```
.controller('ChatsCtrl', ['$scope', '$rootScope', 'UserFactory','Utils', '$ionicPlatfo
rm', '$state', function($scope, $rootScope,UserFactory, Utils, $ionicPlatform, $state)
 {
    $ionicPlatform.ready(function() {
        $scope.$on('$ionicView.enter', function(scopes, states) {
            var olUsers = UserFactory.getOLUsers();

            $scope.chatHistory = [];
            $scope.$on('AddNewChatHistory', function() {
                var ch = $rootScope.chatHistory,
                matchedUser;
                for (var i = 0; i < ch.length; i++) {
                    for (var j = 0; j < olUsers.length; j++) {
                        if (Utils.escapeEmailAddress(olUsers[j].email) == ch[i]) {
                            matchedUser = olUsers[j];
                        }
                    };

                    if (matchedUser) {
                     $scope.chatHistory.push(matchedUser);
                    } else {
                        $scope.chatHistory.push({
                            email: Utils.unescapeEmailAddress(ch[i]),
                            name: 'User Offline'
                        })
                    }
                };
            });

            $scope.redir = function(user) {
                $state.go('chat-detail', {
                    otherUser: user
                });
            }

            $rootScope.$on('newChatHistory', function($event) {
                $scope.$broadcast('AddNewChatHistory');
            });

            $scope.$broadcast('AddNewChatHistory');
        })
    });
}])
```

> 注意看*matchedUser*对象。在用户有聊天历史但是用户在离线状态的时候，这个值将会设为一个离线值，之前也说过。

在这里，我们一直监听从*run*方法里面广播的*newChatHistory*事件。最后，当当前登录用户点击用户名的时候，我们将应用重定向到*chat-detail*视图。

接下来是应用里最活跃的页面，这个页面用来与其他用户交互。他的控制器代码太多，没法

一次贴出来。因此，我将控制器分离成不同的逻辑块，逐个解释。
首先，我们添加了控制器定义和他的依赖：

```
.controller('ChatDetailCtrl', ['$scope', 'Loader','$ionicPlatform', '$stateParams', 'U
serFactory', 'FBFactory',
'$ionicScrollDelegate', '$cordovaImagePicker', 'Utils','$timeout', '$ionicActionSheet'
, '$cordovaCapture',
'$cordovaGeolocation', '$ionicModal',function($scope, Loader, $ionicPlatform, $statePa
rams,UserFactory, FBFactory, $ionicScrollDelegate, $cordovaImagePicker,
Utils, $timeout, $ionicActionSheet, $cordovaCapture,$cordovaGeolocation, $ionicModal)
{
    $ionicPlatform.ready(function() {
        Loader.show('Establishing Connection...');
        // controller code here..
    });
}])
```

好多依赖库哇！
接下来获取*chatToUser*然后将他添加到scope。完成之后，我们连接到动态Firebase终端。以下代码（将会在结束前）将会添加到*ChatDetailCtrl*中：

```
$scope.chatToUser = $stateParams.otherUser;
$scope.chatToUser = JSON.parse($scope.chatToUser);
$scope.user = UserFactory.getUser();

$scope.messages = FBFactory.chatRef($scope.user.email,
$scope.chatToUser.email);
$scope.messages.$loaded().then(function() {
    Loader.hide();
    $ionicScrollDelegate.scrollBottom(true);
});
```

我们使用*$ionicScrollDelegate*服务来滚动视图面板到最后一条聊天信息。接下来，新建一个方法用作向Firebase添加新的聊天信息：

```
function postMessage(msg, type, map) {
    var d = new Date();
    d = d.toLocaleTimeString().replace(/:\d+ /, ' ');
    map = map || null;
    $scope.messages.$add({
        content: msg,
        time: d,
        type: type,
        from: $scope.user.email,
        map: map
    });

    $scope.chatMsg = '';
    $ionicScrollDelegate.scrollBottom(true);
}
```

当用户输入文本点后点击**Send**的时候，我们调用*sendMessage*方法：

```
$scope.sendMessage = function() {
    if (!$scope.chatMsg) return;
    var msg = '<p>' + $scope.user.cachedUserProfile.name + ' says : <br/>' + $scope.ch
atMsg + '</p>';
    var type = 'text';
    postMessage(msg, type);
}
```

使用以下Action Sheet（动作表单）服务，来展示一系列的列表，例如：*Share Picture*分享图片,*Take Picture*拍照,*Share My Location*分享位置信息：

```
$scope.showActionSheet = function() {
var hideSheet = $ionicActionSheet.show({
buttons: [{
    text: 'Share Picture'
}, {
    text: 'Take Picture'
}, {
    text: 'Share My Location'
}],
cancelText: 'Cancel',
cancel: function() {
    // add cancel code..
    Loader.hide();
},
buttonClicked: function(index) {
    // Clicked on Share Picture
    if (index === 0) {
        Loader.show('Processing...');
        var options = {
            maximumImagesCount: 1
```

```
        };
        $cordovaImagePicker.getPictures(options)
          .then(function(results) {
            if (results.length > 0) {
                var imageData = results[0];
                Utils.getBase64ImageFromInput(
                imageData, function(err, base64Img)
                {
                //Process the image string.
                postMessage('<p>' + $scope.user.cachedUserProfile.name + ' posted : <b
r/><img class="chat-img" src="' + base64Img + '">', 'img');
                Loader.hide();
                });
            }
        }, function(error) {
            // error getting photos
            console.log('error', error);
            Loader.hide();
        });
    }
    // Clicked on Take Picture
    else if (index === 1) {
        Loader.show('Processing...');
        var options = {
            limit: 1
        };
        $cordovaCapture.captureImage(options).then(function(imageData) {
            Utils.getBase64ImageFromInput(imageData[0].fullPath, function(err, base64I
mg) {
                //Process the image string.
                postMessage('<p>' + $scope.user.cachedUserProfile.name + ' posted : <b
r/><img class="chat-img" src="'+ base64Img + '">', 'img');
                Loader.hide();
                });
        }, function(err) {
            console.log(err);
            Loader.hide();
        });
    }
    // clicked on Share my location
    else if (index === 2) {
        $ionicModal.fromTemplateUrl('templates/map-modal.html', {
            scope: $scope,
            animation: 'slide-in-up'
        }).then(function(modal) {
            $scope.modal = modal;
            $scope.modal.show();
            $timeout(function() {
                $scope.centerOnMe();
            }, 2000);
        });
    }
    return true;
```

```
}
});
}
```

当选中Action Sheet中的选项的时候，如果：

- index = 0：我们调用*$cordovaImagePicker*服务让用户选择图片。当用户选中图片后，我们调用*Utils.getBase64ImageFromInput*方法将图片转变成base64的字符串。 然后我们使用*postMessage*方法将他发送给Firebase。
- index = 1：我们调用*$cordovaCapture*服务的*captureImage*方法来照相，转成base64字符串，然后发送给Firebase。
- index = 2：我们调用Ionic Modal带有地图，指向用户当前位置。

为了在弹出框中使用地图，我们需要在scope上定义一些方法：

```
$scope.mapCreated = function(map) {
    $scope.map = map;
};

$scope.closeModal = function() {
    $scope.modal.hide();
};

$scope.centerOnMe = function() {
    if (!$scope.map) {
        return;
    }
    Loader.show('Getting current location...');
    var posOptions = {
        timeout: 10000,
        enableHighAccuracy: false
    };

    $cordovaGeolocation.getCurrentPosition(posOptions).
      then(function(pos) {
        $scope.user.pos = {
            lat: pos.coords.latitude,
            lon: pos.coords.longitude
        };

        $scope.map.setCenter(new google.maps.LatLng($scope.user.pos.lat, $scope.user.p
os.lon));
        $scope.map.__setMarker($scope.map,  $scope.user.pos.lat, $scope.user.pos.lon);
        Loader.hide();
    }, function(error) {
        alert('Unable to get location, please enable  GPS to continue');
        Loader.hide();
        $scope.modal.hide();
    });
};

$scope.selectLocation = function() {
    var pos = $scope.user.pos;

    var map = {
        lat: pos.lat,
        lon: pos.lon
    };
    var type = 'geo';

    postMessage('<p>' + $scope.user.cachedUserProfile.name + ' shared : <br/>', type,
map);
    $scope.modal.hide();
}
```

地图创建的时候将会调用*mapCreated*方法。*closeModal*将用来关闭弹出框。*centerOnMe*方法在地图初始化完成之后自动调用。这个方法使用*$cordovaGeolocation.getCurrentPosition*来获取用户当前位置。 一旦位置获取成功，他会用一个标记来替换那个点。如果没法通过*$cordovaGeolocation.getCurrentPosition*来获取用户坐标，我们就会询问用户激活GPS。
当功能正常运行的时候，就能更好的理解了。
最后，*AccountCtrl*，连接到*Tab3*。这个控制器有管理用户登出的方法：

```
.controller('AccountCtrl', ['$scope', 'FBFactory', 'UserFactory','$state',
    function($scope, FBFactory, UserFactory, $state) {

        $scope.logout = function() {
            FBFactory.auth().$unauth();
            UserFactory.cleanUser();
            UserFactory.cleanOLUsers();

            // remove presence
            var onlineUsers = UserFactory.getOLUsers();
            if (onlineUsers && onlineUsers.$getRecord) {
                var presenceId = UserFactory.getPresenceId();
                var user = onlineUsers.$getRecord();
                onlineUsers.$remove(user);
            }

            UserFactory.cleanPresenceId();
            $state.go('main');
        }
    }
]);
```

> 在此，你也可以设置一些个人偏好，例如当打开应用的时候显示通知，播放声音等等。

设置模板

现在我们完成了所有的Javascript代码，我们接下来就是实现每个视图的模板来。所有视图都是逻辑实现，简单易懂。 第一个是*main.html*，*www/templates/main.html*的全部内容：

```
<ion-view view-title="IONIC CHAT APP" cache-view="false">
    <ion-content>
        <ion-slide-box does-continue="true" auto-play="true" showpager="false">
            <ion-slide>
                <label class="t-r">Share Photos seamlessly between family & Friends</l
abel>
                <img src="http://placeimg.com/640/480/tech/grayscale" />
            </ion-slide>
            <ion-slide>
                <label class="c-c">Simple One click login to start the fun!!</label>
                <img src="http://placeimg.com/640/480/people/sepia" />
            </ion-slide>
            <ion-slide>
                <label class="b-r">Notify people where you are with one click location
 sender</label>
                <img src="http://placeimg.com/640/480/tech/sepia" />
            </ion-slide>
        </ion-slide-box>
        <div class="text-center padding">
            <h3>The Super chat app, lets you connect with your friends, share images,
audio, video, geo-location and ofcourse texts!</h3>
            <button class="button button-dark" ngclick="loginWithGoogle()">
            Login With Google
            </button>
        </div>
    </ion-content>
</ion-view>
```

接下来是*www/templates/tabs.html*文件。使用以下内容替换他的全部内容：

```
<ion-tabs class="tabs-striped tabs-top tabs-background-positive
tabs-color-light">
    <!-- Dashboard Tab -->
    <ion-tab title="IONIC CHAT APP" icon-off="ion-ios-pulse" iconon="ion-ios-pulse-str
ong" href="#/tab/dash">
        <ion-nav-view name="tab-dash"></ion-nav-view>
    </ion-tab>
    <!-- Chats Tab -->
    <ion-tab title="IONIC CHAT APP" icon-off="ion-ios-chatboxesoutline" icon-on="ion-i
os-chatboxes" href="#/tab/chats">
        <ion-nav-view name="tab-chats"></ion-nav-view>
    </ion-tab>
    <!-- Account Tab -->
    <ion-tab title="IONIC CHAT APP" icon-off="ion-ios-gear-outline" icon-on="ion-ios-g
ear" href="#/tab/account">
        <ion-nav-view name="tab-account"></ion-nav-view>
    </ion-tab>
</ion-tabs>
```

接下来是*www/templates/tab-dash.html*：

```
<ion-view view-title="IONIC CHAT APP">
    <ion-content>
        <ion-list>
            <ion-item ng-show="users.length == 1">
                <h3 class="text-center padding">Looks like no one is online</h3>
            </ion-item>
            <ion-item class="item-avatar item-icon-right" ngrepeat="user in users | fi
lter:search:user" ng-if="user.email != currUser.email" ng-click="redir('{{user}}')">
                <img ng-src="{{user.picture}}">
                <h2>{{user.name}}</h2>
                <p>{{user.email}}</p>
                <i class="icon ion-chevron-right iconaccessory"></i>
            </ion-item>
        </ion-list>
    </ion-content>
</ion-view>
```

接着*www/templates/tab-chats.html*：

```
<ion-view view-title="IONIC CHAT APP">
    <ion-content>
        <ion-list>
            <ion-item ng-show="chatHistory.length == 0">
                <h3 class="text-center padding">Looks like there is no chat history</h
3>
            </ion-item>
            <ion-item class="item-icon-right item-icon-left" ngclass="{'item-avatar' :
 user.picture}" ng-repeat="user in chatHistory | filter:search:user" ng-if="user.email
 != currUser.email" ng-click="redir('{{user}}')">
                <img ng-src="{{user.picture}}" ngshow="user.picture">
                <h2>{{user.name}}</h2>
                <p>{{user.email}}</p>
                <i class="icon ion-chevron-right iconaccessory"></i>
            </ion-item>
        </ion-list>
    </ion-content>
</ion-view>
```

第三个标签页*www/templates/tab-account.html*：

```
<ion-view view-title="IONIC CHAT APP">
    <ion-content has-header="true">
        <ion-list>
            <!-- Uncomment below if you would like to add preferences to the app -->
            <!-- <ion-item>
<ion-toggle ng-change="updatePreference()" ngmodel="preference.notification" toggle-cl
ass="toggle-positive">Show Notifications</ion-toggle>
</ion-item> -->
            <ion-item>
                <button class="button button-dark button-block" ng-click="logout()">
                Logout
                </button>
            </ion-item>
        </ion-list>
    </ion-content>
</ion-view>
```

所以行为发生的地方*www/templates/chat-detail.html*：

```
<ion-view view-title="{{chatToUser.name}}">
    <ion-pane>
        <ion-content class="has-header padding">
            <div class="button-bar">
                <a class="button button-calm" uisref="tab.dash">Online Users</a>
                <a class="button button-calm" uisref="tab.chats">Chat History</a>
            </div>
            <br>
            <ion-list>
                <ion-item ng-show="messages.length == 0">
                    <h3 class="text-center">No messages yet!</h3>
                </ion-item>
                <ion-item class="item-avatar" ng-repeat="message in messages" ng-class
="{left : message.from == user.email, right : message.from != user.email}">
                    <img ng-src="{{user.cachedUserProfile.picture}}" ng-if="message.fr
om == user.email">
                    <img ng-src="{{chatToUser.picture}}" ngif="message.from != user.em
ail">
                    <p ng-bind-html="message.content"></p>
                    <map inline="true" class="inline-map" lat="{{message.map.lat}}" lo
n="{{message.map.lon}}" ngif="message.map.lat && message.map.lon">
                </ion-item>
                <div class="padding-bottom"></div>
            </ion-list>
        </ion-content>
        <ion-footer-bar class="bar-footer">
            <input class="footerInput" type="text" placeholder="Enter Message" ng-mode
l="chatMsg">
            <button class="button button-dark icon-left ionpaper-airplane" ng-click="s
endMessage();"></button>
            <button class="button button-dark icon-left ion-more" ng-click="showAction
Sheet();"></button>
        </ion-footer-bar>
    </ion-pane>
</ion-view>
```

> 注意在这里我们使用了map指令，内联属性设为*true*

当用户选择分享位置信息的时候，我们需要一个地图展示。现在我们就来创建这个。在*www/templates*中新建一个文件名为*map-modal.html*，更新如下：

```
<ion-modal-view>
    <ion-content scroll="false">
        <map on-create="mapCreated(map)"></map>
    </ion-content>
    <ion-footer-bar class="bar-stable">
        <a ng-click="selectLocation()" class="button button-icon icon ion-checkmark">S
hare</a>
        <a ng-click="closeModal()" class="button button-icon icon ion-close">Cancel</a
>
    </ion-footer-bar>
</ion-modal-view>
```

设置SCSS

为完善此应用，还需要对*scss/ionic.app.scss*做如下变更。
首先，重写4个Ionic SCSS变量，然后包含Ionic SCSS框架：

```
$positive: #1976D2 !default;
$font-family-base: 'Lato',
sans-serif !default;
$tabs-striped-off-opacity: 1 !default;

// The path for our ionicons font files, relative to the built CSS in www/css
$ionicons-font-path: "../lib/ionic/fonts" !default;

// Include all of Ionic
@import "www/lib/ionic/scss/ionic";
```

接下来，覆盖标题栏和滑动框的样式：

```
.bar .title {
    font-size: 21px;
}

.slider {
    background-color: #eee;
    min-height: 200px;
    max-height: 400px;
}

ion-slide img {
    width: 100%;
    height: 50%;
    margin: 0 auto;
    display: block;
    max-height: 350px;
    max-width: 500px;
}

.t-r {
    position: absolute;
    top: 5px;
    right: 0;
    margin: 20px;
    margin-right: 5px;
    margin-left: 25px;
    text-align: center;
    width: 84%;
}
```

```
.b-r {
    position: absolute;
    bottom: 5px;
    margin: 20px;
    right: 0px;
    margin-right: 5px;
    margin-left: 25px;
    text-align: center;
    width: 84%;
}

.c-c {
    position: absolute;
    top: 25%;
    margin: 20px;
    left: 0px;
    margin-right: 5px;
    margin-left: 25px;
    text-align: center;
    width: 84%;
}

ion-slide label {
    font-size: 21px;
    color: #333;
    padding: 5px;
    border-radius: 5px;
    background: linear-gradient(to right, #e2e2e2 0%, #dbdbdb 50%,#d1d1d1 51%, #fefefe
 100%);
    opacity: 0.8;
}
```

为聊天界面添加以下样式：

```
.footerInput {
    width: 77%;
}

.chat-img {
    width: 50%;
}

.left,
.right {
    width: 75%;
    clear: both;
    margin: 5px;
}

.left {
    float: left;
    text-align: left;
}

.right {
    float: right;
    text-align: right;
}

.usr-img {
    width: 48px;
}

map {
    display: block;
    width: 100%;
    height: 100%;
}

.inline-map {
    height: 200px;
    border: 1px solid #787878;
}

.scroll {
    height: 100%;
}
```

测试应用

现在，应用创建完成，我们需要添加iOS平台和Android平台来进行测试了：

```
ionic platform add ios
ionic platform add android
```

接下来，我们就要模拟/运行app了。我有一个三星Galaxy Note 3和一个iOS模拟器作为测试设备。
我在Android上运行，在iOS模拟器上模拟过来。你也可以同Android模拟器和iOS模拟器进行测试。 利用如下命令启动或者模拟应用：

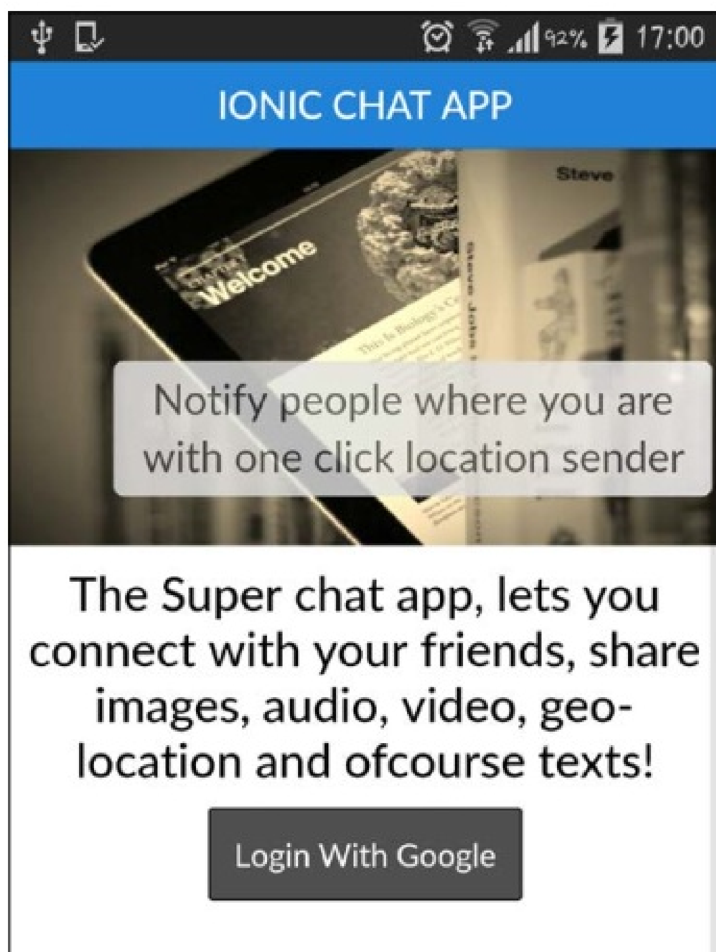
```
ionic run android -l -c
```

也可以这么用：

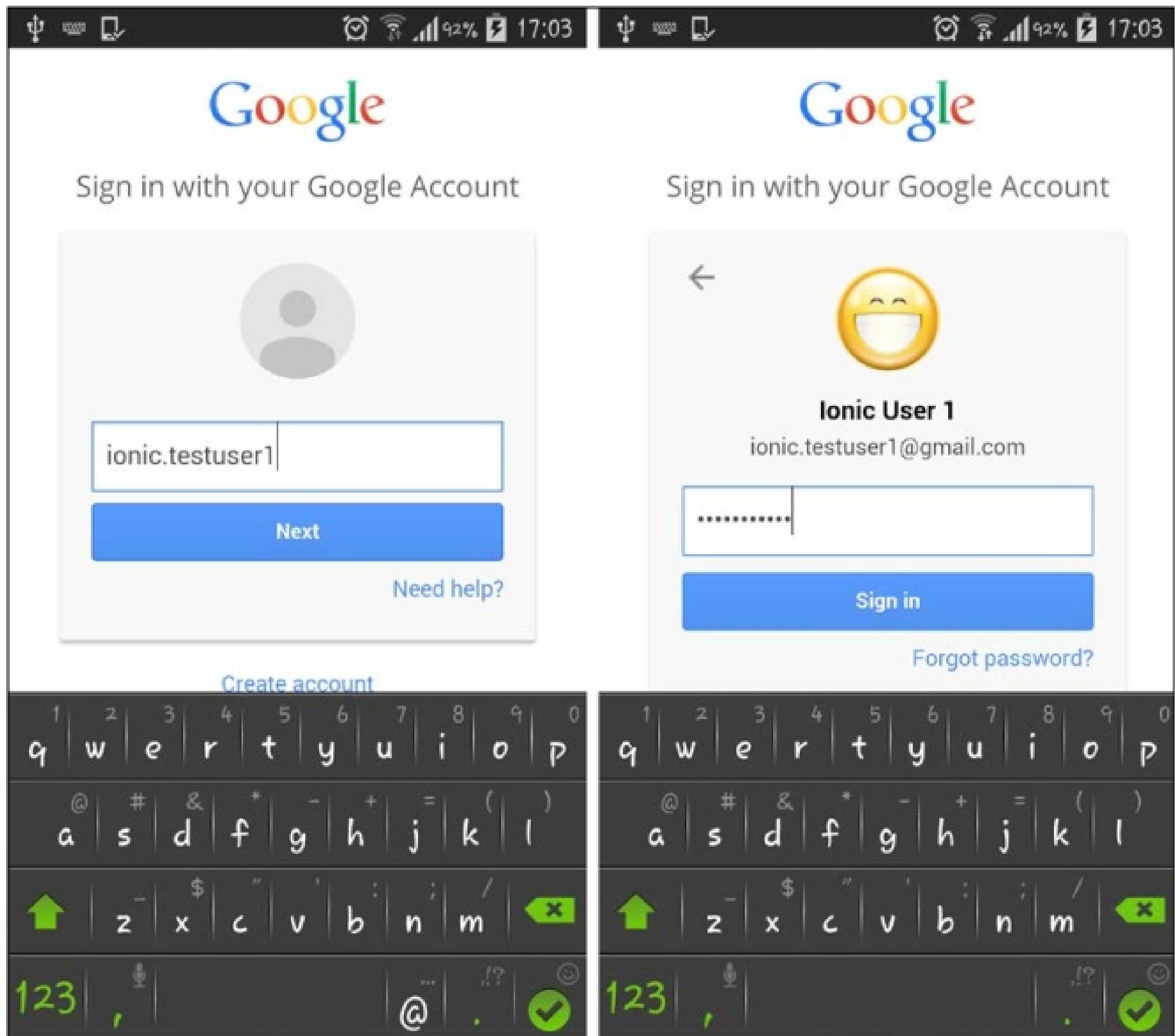
```
ionic emulate ios –l –c
```

> **-l** 标记设置在模拟或者运行中实时重新加载，**-c** 标记激活JavaScript控制台日志输出到命令行或者终端。在模拟器和设备上调试Ionic应用，这两个算是最有用的标记了。

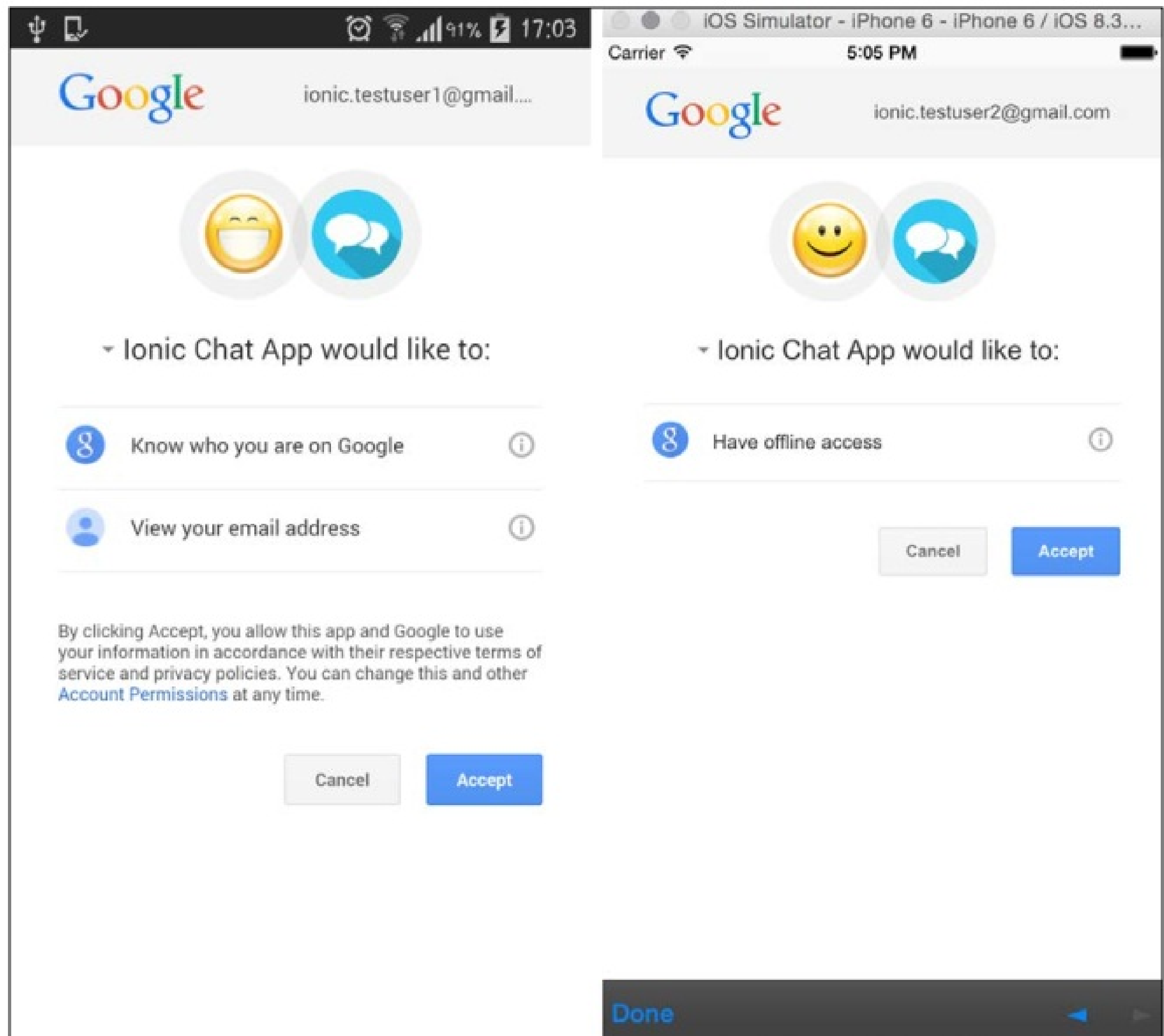
应用运行成功之前，主页大概是这样的：



点击**Login With Google**的时候，会看到Google授权页如下：

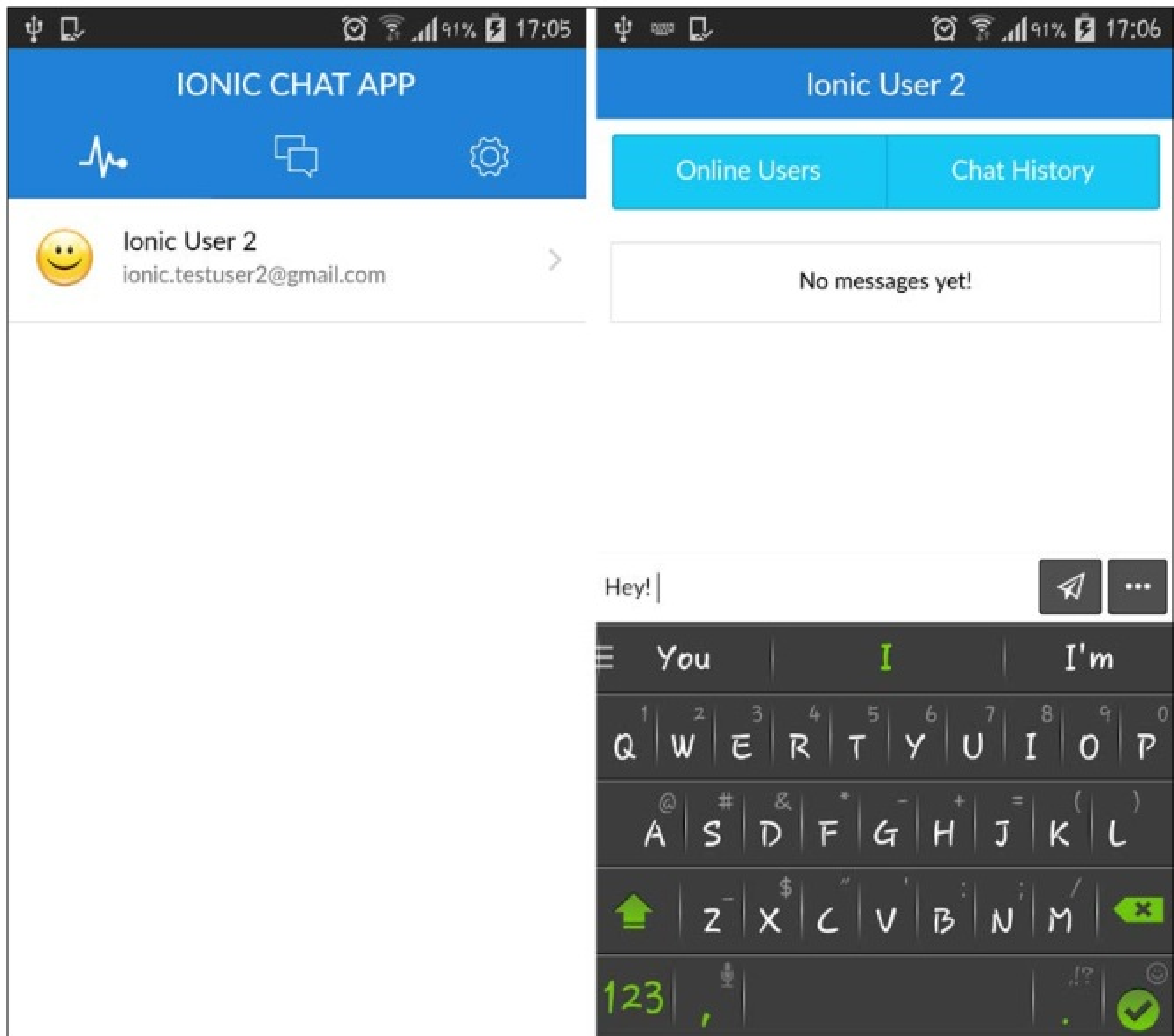


授权认证成功之后，将会看到许可界面（下面截屏的左边），如果你是回流用户的话，将会问你离线访问（下面截屏右边）：

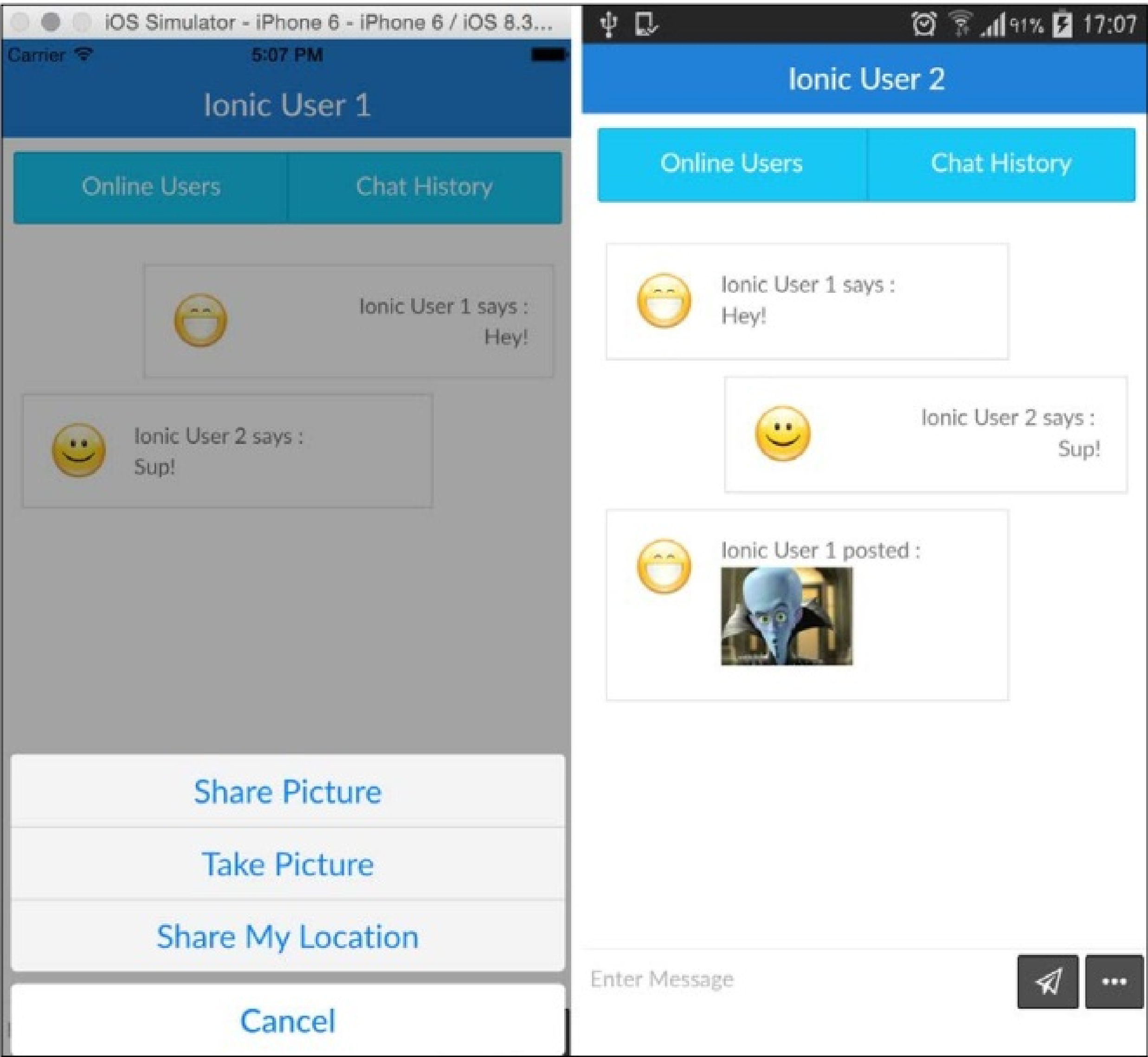


> 在Android设备上是使用*ionic.testuse1@gmail.com*登录的，在iOS模拟器上使用的是*ionic.testuser2@gmail.com*登录的。

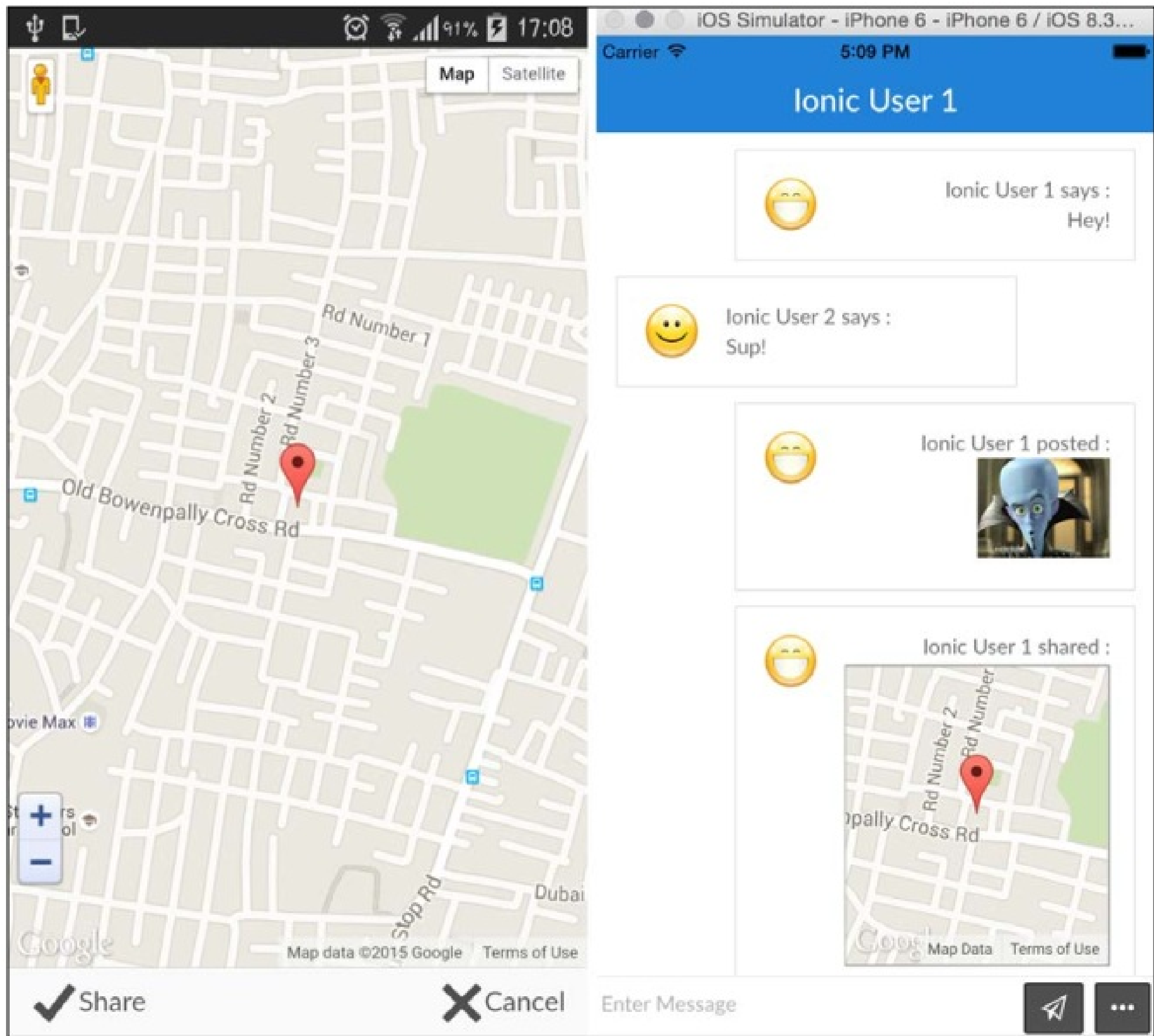
成功登录之后，将会显示用来展示离线用户的dashboard标签页，如下左。当点击一个用户，将被带到聊天界面，如下右：



用户可以通过在文本域中输入信息然后点击图标按钮来发送信息。然后，通过点击more图标按钮来显示一个动作表单如下左。 可以通过动作表单行为来与其他用户分享图片，如下右：


iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.3...
Carrier
5:07 PM
Ionic User 1
Online Users
Chat History
Ionic User 1 says :
Hey!
Ionic User 2 says :
Sup!
Share Picture
Take Picture
Share My Location
Cancel
17:07
Ionic User 2
Online Users
Chat History
Ionic User 1 says :
Hey!
Ionic User 2 says :
Sup!
Ionic User 1 posted :
Enter Message

也可以选择**Share My Location**选项（如下左）来分享地理信息。其他用户可以在聊天界面里面看到（如下右）：



这就是我们的聊天应用！

你可以在任何时间点访问Firebase forge来查看存储的数据。效果大概如下：



总结

本章中，我们学会了如何使用Ionic，Cordova以及Firebase来制作一个简单的聊天应用。我们先从理解结构开始，之后学习了Firebase和AngularFire，然后我们将它们统统整合到Ionic应用中。 同时我们也学习了一些核心思想的实现，例如在Ionic应用中整合Cordova插件和根据需求使用不同功能。

下一章是最后一张，我们将学习如何生成如此应用的设备指定安装包，如何与世界分享他。

本章中，我们将学习三种生成Ionic应用安装包的方法：一是使用PhoneGap构建服务，二是使用Cordova CLI，第三个是使用Ionic package服务。 我们同时会为Android和iOS操作系统生成安装包。本章涵盖主题如下：

- 生成图标和预览图
- 验证config.xml
- 使用PhoneGap服务生成安装包
- 使用Cordova CLI生成安装包
- 使用Ionic package服务生成安装包

预备应用的分发

现在我们成的创建了我们的Ionic应用，我们想要分发出去。接触大量用户的最佳方法是App Store。 但是在分发此应用之前，我们需要为应用制作特定的图标和截屏。截屏是可选的，根据产品理念而定。

设置图标也截屏

默认，在运行：

```
ionic platform add android
```

或者：

```
ionic platform add ios
```

的时候，CLI会自动添加一个文件夹，名为*resources*。可以在第八章 制作一个聊天*App*中创建的*ionc-chat-app*中查看。 *resources*文件夹里面有Ionic子文件夹，Android子文件夹。他是根据你添加的平台生成的。这些文件夹内都是有两个子文件夹组成：icon 和 splash。

> 如果你的应用要使用截屏的话，那么需要保留*splash*文件夹；如果不需要的话，删除此文件夹可以为安装包节省几个字节

为生成图标，可以将你的图标做成尺寸大于1024*1024，然后利用以下任一服务：

- http://icon.angrymarmot.org/
- http://makeappicon.com/
- http://www.appiconsizes.com/

这些服务都是用来为Android和iOS生成图标和截屏的。

> 我跟这上面的服务没有任何关系。所以使用过程中出现任何风险，本人概不负责。

另外，你可以将*icon.png*和*splash.png*放到*resources*文件夹内然后运行：

```
ionic resources
```

Ionic将负责把你的图片上传到云，然后根据需求调整尺寸，然后将调整好尺寸的返回的图片保存到*resources*文件夹内。 如果你只想调整图标，使用如下命令：

```
ionic resources --icon
```

如果只想调整截屏，使用如下命令：

```
ionic resources --splash
```

> 你可以使用这个psd来设计你的图标：http://code.ionicframework.com/resources/icon.psd 可以使用这个psd来设计截屏：http://code.ionicframework.com/resources/splash.psd

更新config.xml文件

我们已经知道，*config.xml*是Cordova API在生成平台特定安装包的时候唯一信任的真理。 因此，在开始部署之前一定要验证好此文件。可以根据以下检查列表来检查是否正常：

- Widget ID是否定义与有效
- widget 版本是否定义且有效
- 未防应用更新，更新widget版本并验证是否有效
- name标签已定义且有效
- description已定义且有效
- 作者（Author）信息已定义且有效
- Access标签已定义且限制到所需的域名：https://github.com/apache/cordova-plugin-whitelist#network-requestwhitelist
- 允许跳转的连接已定义好且限制到所需的域名：https://github.com/apache/cordova-plugin-whitelist#intent-whitelist
- 交叉检查（Cross=check）偏好设置
- 交叉检查图标和截屏路径
- 交叉检查许可（如果有的话）
- 更新*index.html*的内容安全策略元标签https://github.com/apache/cordova-plugin-whitelist#content-securitypolicy

验证完上面的点之后，我们就可以开始生成安装包了。

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通: https://github.com/learning-ionic/Chapter-9

PhoneGap服务

第一个探索的应用安装包生成方式是使用PhoneGap构建服务。这应该是最简单的Android和iOS安装包生成方式了。

流程非常简单。我们将整个项目上传到PhoneGap构建服务他就会负责构建安装包了。

> 如果你觉得上传整个项目不切实际，你可以只上传*www*文件夹；但是，需要做以下变更： 首先，将*config.xml*移动到*www*文件夹内。 接着，将*resources*文件夹移动到*www*文件夹内。 最后，修改*config.xml*里面的*resources*文件夹的路径。 如果你发现你经常干这件事情的话，建议你写一个脚本来生成一个*PhoneGap deployable*（可部署）文件夹，用来容纳变更后的项目。

如果你打算只发布Android版本，你就不用再做别的操作了。 如果你打算生成iOS安装包，那么你就需要一个Apple Developer Account（苹果开发者账号）并遵照步骤生成所需证书：http://docs.build.phonegap.com/en_US/signing_signing-ios.md.html

> 你也可以按照这里来给对你的Android应用进行签名：http://docs.build.phonegap.com/en_US/signing_signing-android.md.html

一旦得到证书和密钥，你就可以开始生成安装包里。按照以下步骤：

1. 新建一个PhoneGap账号并登录：https://build.phonegap.com/plans
2. 接下来，导航到 https://build.phonegap.com/people/edit ，选择Siging Keys标签页，上传iOS和Android证书
3. 然后，导航到 https://build.phonegap.com/apps ，点击*New App*。 作为免费计划的一部分，在他们还从Pulic Git资源目录拉取的时候，你可以拥有任意多数量的app。 你也可以通过Private repo（资源目录）或者上传一个ZIP文件来创建一个私有应用。
4. 为测试此服务，我们来创建一个*.zip*文件（不是*.rar*或者*.7z*），遵照如下文件夹结构：
    - App（根文件夹） config.xml resources（文件夹） www（文件夹） 然后，如前所述，更新config.xml
5. 上传ZIP文件到 https://build.phonegap.com/apps 点击 Create app

将花费数分钟来完成生成流程。

> 有时候，构建服务会出现错误。等待之后再重试。根据构建服务器的负载，构建流程有时候会比预想的更长。

使用Cordova CLI生成安装包

现在，我们将使用Cordova CLI来生成Android和iOS安装包。

Android安装包

首先来看Android安装包的生成。步骤如下：

1. 在项目的根目录下打开终端/命令行
2. 移除不需要的插件： **ionic plugin rm cordova-plugin-console**
3. 以发布模式构建应用： **cordova build --release android** 。 将会在/platforms/android/build/outputs/apk/android-release-unsigned.apk生成一个未签名的安装包
4. 接下来，我们需要制作一个签名密钥。如果已经有了的话或者你是更新一个已有的app的话，可以直接进行到第六步
5. 使用密钥工具生成私有密钥。 创建一个文件夹名为*deploy-key*，所有的密钥都将存放在此。 文件夹创建好了之后，通过*cd*命令进入到此文件夹，然后运行：
   keytool -genkey -v -keystore app-name-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000
   然后你将被问到如下问题，你可以如下回答：

```
→ deploy-keys keytool -genkey -v -keystore app-name-release-key.keystore -alias my-ionic-app -keyalg RSA -
keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Arvind Ravulavaru
What is the name of your organizational unit?
  [Unknown]:  Stack Engineering
What is the name of your organization?
  [Unknown]:  JackalStack Technologies Pvt. Ltd.
What is the name of your City or Locality?
  [Unknown]:  Hyderabad, India
What is the name of your State or Province?
  [Unknown]:  Andhra Pradesh
What is the two-letter country code for this unit?
  [Unknown]:  IN
Is CN=Arvind Ravulavaru, OU=Stack Engineering, O=JackalStack Technologies Pvt. Ltd., L="Hyderabad, India", S
T=Andhra Pradesh, C=IN correct?
  [no]:  YES

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=Arvind Ravulavaru, OU=Stack Engineering, O=JackalStack Technologies Pvt. Ltd., L="Hyderabad,
 India", ST=Andhra Pradesh, C=IN
Enter key password for <my-ionic-app>
        (RETURN if same as keystore password):
[Storing app-name-release-key.keystore]
```

> 如果你丢失了此文件或者忘记别名或者密码的话，你将永远不能像app store提交更新了，永远。

6. 可选步骤：你可以将*android-release-unsigned.apk*拷贝到*deploy-key*文件夹，然后运行在其中运行下面的命令。但是，我还是他这些文件留在他们原先的位置。

7. 接下来，使用*jarsigner*工具，给未签名的APK签名： **jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 - keystore app-name-release-key.keystore ../platforms/android/build/outputs/apk/android-releaseunsigned.apk my-ionic-app** 这个过程将会问到你密码，也就是创建keystore第一步输入的。一旦签名流程完成，*android-release-unsigned.apk*将被同名的签名版替换掉。

    > 以上命令在*deploy-keys*文件夹内运行

8. 最后，我们运行*zipalign*来优化APK。 **zipalign -v 4 ../platforms/android/build/outputs/apk/androidrelease-unsigned.apk my-ionic-app.apk**

以上命令将会在*deploy-keys*里面创建一个*my-ionic-app.apk*。
现在，你可以将APK投放到app store了。

iOS安装包

接下来，我们将使用Xcode来为iOS生成安装包。执行如下步骤即可：

1. 在项目根目录下打开命令行/终端
2. 移除不需要的插件： **ionic plugin rm cordova-plugin-console**
3. 运行： **ionic build –release ios**
4. 导航到 *platform/ios* 文件夹，使用Xcode启动*projectname.xcodeproj*
5. 一旦项目导入Xcode完成，选择了**iOS Device**之后，从导航菜单中选择**Product**，然后**Archive**

    > 如果**Archive**选项没有激活，参考：http://stackoverflow.com/a/18791703

6. 接下来，在导航菜单中选择**Window**然后选择**Organizer**。你将会看到一系列创建好的结构体。

7. 点击你现在已经创建好的结构体的缩略图，点击**Submit to App Store**。将会验证你的账户然后应用将上传到Apple Store。
8. 最后，登录Apple Store设置截屏，描述，等等。

Ionic package

编写本书的时候，Ionic package任务还是beta版。因此，我最后才讲他。

将项目上传到Ionic云

使用Ionic云服务生成安装包非常简单。首先，使用如下命令将应用上传到我们的Ionic账号：

```
ionic upload
```

> 在执行上面的命令之前先登录Ionic账号。如果你的项目有敏感信息的话，在将应用上传到云服务之前与Ionic许可证交叉对比

一旦应用上传成功，将会为你的应用生成一个app ID。你可以在项目根目录下的*ionic.project*中找到app ID。

生成所需密钥

按照Android安装包部分的第五步，生成*keystore*文件。
接下来，我们使用*ionic package*命令来生成安装包，如下：

```
ionic package <options> [debug | release] [ios | android]
```

有如下可用选项：

```
package [options] <MODE> <PLATFORM> .......  Package an app using the Ionic Build service (beta)
                                             <MODE> "debug" or "release"
                                             <PLATFORM> "ios" or "android"
        [--android-keystore-file|-k]  .....  Android keystore file
        [--android-keystore-alias|-a]  ....  Android keystore alias
        [--android-keystore-password|-w]  .  Android keystore password
        [--android-key-password|-r]  ......  Android key password
        [--ios-certificate-file|-c]  ......  iOS certificate file
        [--ios-certificate-password|-d]  ..  iOS certificate password
        [--ios-profile-file|-f]  ..........  iOS profile file
        [--output|-o]  ....................  Path to save the packaged app
        [--no-email|-n]  ..................  Do not send a build package email
        [--clear-signing|-l]  .............  Clear out all signing data from Ionic server
        [--email|-e]  .....................  Ionic account email
        [--password|-p]  ..................  Ionic account password
```

例如，如果你想生成一个Android发布版：

```
ionic package release android -k app-name-release-key.keystore -a myionic-app -w 12345
678 -r 12345678 -o ./ -e arvind.ravulavaru@gmail.com -p 12345678
```

> 我们在*deploy-keys*文件夹内运行此命令

同样，对于iOS：

```
ionic package release ios -c certificate-file -d password -f profilefile -o ./ -e arvi
nd.ravulavaru@gmail.com -p 12345678
```

> *ionic package*命令在Ionic CLI 1.5.2中已经移除。相关信息参考：
> https://github.com/driftyco/ionic-cli/issues/214#issuecomment-109349399

总结

完成这些之后，我们的Ionic之旅就正式结束了。快速总结一下，我们从了解为何使用AngularJS开始。 然后，我们学习了移动混合应用如何运行，Cordova和Ionic可以用在哪里。接下来，我们学习了大量的Ionic模板然后快速浏览了Ionic CSS组件，Ionic指令以及服务。我们利用这些知识为一个secure REST API制作了一个Ionic客户端。 接着，我们学习了Cordova和ngCordova，以及如何使用它们。 我们利用Ionic与Cordova的长处制作了一个聊天应用。 最后，我们学习如何为制作好的应用生成安装包并发布到app store。

附加主题与贴士

本书的主要目的是让你尽可能的熟悉Ionic。所以，从第一章到第九章，我从Cordova，AngularJS，Ionic基本知识开始，逐步深入。

在此附录中，我们将探索其他一些Ionic CLI：ionc.io， ionic-box 和 Sublime Text 插件

> 关于本章，你也可以通过以下Github目录来访问源代码，发起issue，与作者沟通: https://github.com/learning-ionic/Appendix

Ionic CLI

Ionic CLI越来越强大了。写作此书的时候，最新的Ionic CLI版本是1.5.5。本书使用的是Ionic 1.5.0.

Ionic登录

有三种方法登录ionic云服务。
第一种，有提醒的：

```
ionic login
```

第二种，没有提醒的：

```
ionic login --email arvind.ravulavaru@gmail.com --password 12345678
```

最后，使用环境变量。可以设置*IONIC_EMAIL*和*IONIC_PASSWORD*环境变量，Ionic CLI会不用任何提醒，自己去获取。 但是这么做很不安全，因为密码赤裸裸的展示为普通文本。

> 首先你的有一个Ionic.io账号，否则登录不会成功的

Ionic开始任务

首先，我们看一下**No Cordova**标记选项。

No Cordova标记

ionic start 任务是最简单的创建Ionic应用的方法之一。本书中，我们已经多次用过。但是，我们知道，Ionic可以不用Cordova。
想要不依赖Cordova的依赖的话，在使用 *ionic start* 的时候就需要添加一个 *-w* 标记或者 *-no-cordova* 标记：

```
ionic start -a "My Mobile Web App" -i app.web.mymobile -w myMobileWebApp maps
```

生成的项目结构如下：

```
.
└── myMobileWebApp
    ├── bower.json
    ├── gulpfile.js
    ├── ionic.project
    ├── package.json
    ├── scss
    │   └── ionic.app.scss
    └── www
        ├── css
        ├── img
        ├── index.html
        ├── js
        ├── lib
        └── templates
```

接着，和往常一样使用 *cd* 命令，进入 *myMobileWebApp* 运行 *ionic serve* 。

新建项目的时候附加SCSS支持

新建项目默认附加SCSS支持，可以在使用 *ionic start* 的时候，添加 *-s* 或者 *-sass* 标记：

```
ionic start -a "Example 1" -i app.one.example --sass example1 blank
```

列出所有Ionic模板

想要查看所有的可用模板，运行 *ionic start* 的时候添加 *-l* 或者 *--list* 标记：

```
ionic start -l
```

到本书编写的今天为止，有以下可用模板：

```
blank ................ A blank starter project for Ionic
complex-list ......... A complex list starter template
maps ................. An Ionic starter project using Google Maps and a
side menu
salesforce ........... A starter project for Ionic and Salesforce
sidemenu ............. A starting project for Ionic using a side menu
with navigation in the content area
tabs ................. A starting project for Ionic using a simple tabbed
interface
tests ................ A test of different kinds of page navigation
```

> 在编写本书的今天，*complex-list*模板还是个空白模板，*tests*模板是ionic团队内部测试使用的。

App ID

如果你在使用Ionic云服务的话，那么你在云服务上创建的每个项目都会指派一个app ID（具体参考下面的*Ionic.io*应用部分）。 app ID将会保存在根目录下的的*ionic.project*文件里。当你新建一个项目的时候，app ID是空的。如果想将新搭建的项目关联到云端已有的应用上的话， 可以运行*ionic start*附加*--ion-app-id*标记并传入云服务生成的app ID：

```
ionic start -a "Example 2" -i app.two.example --io-app-id "b82348b5" example2 blank
```

此时，*ionic.project*是这样子的：

```
{
    "name": "Example 2",
    "app_id": "b82348b5"
}
```

Ionic link

本地创建的项目可以通过以下命令连接到一个云端项目（具体参考 *Ionic.io*应用部分）：

```
ionic link b82348b
```

也可以通过以下命令移除已有的app ID:

```
ionic link --reset
```

Ionic info

想要查看已安装的依赖包和他们的版本号，运行：

```
ionic info
```

列出的信息应该是这样的：

```
Cordova CLI: 5.0.0
Gulp version:  CLI version 3.8.11
Gulp local:
Ionic Version: 1.0.0
Ionic CLI Version: 1.5.0
Ionic App Lib Version: 0.1.0
ios-deploy version: 1.7.0
ios-sim version: 3.1.1
OS: Mac OS X Yosemite
Node Version: v0.12.2
Xcode version: Xcode 6.3.2 Build version 6D2105
```

Ionic模板

除了可以使用start任务查看可用模板之外，也可以通过templates任务来查看可以模板：

```
ionic templates
```

Ionic browsers

Ionic默认使用操作系统的默认浏览器渲染webview的。 想要获取更好的用户体验或者使用最新特性的话，你可以将默认的浏览器替换为Crosswalk（ https://crosswalkproject.org/ ）或者Crosswalk Lite（ (https://github.com/crosswalk-project/crosswalk-website/wiki/Crosswalk-Project-Lite ）。 目前，只能使用这两个浏览器。可以通过以下命令查看当前支持的浏览器：

```
ionic browser list
```

然后，会看到：

```
iOS - Browsers Listing:
Not Available Yet - WKWebView
Not Available Yet - UIWebView


Android - Browsers Listing:
Available - Crosswalk - ionic browser add crosswalk
        Version 8.37.189.14
        Version 9.38.208.10
        Version 10.39.235.15
        Version 11.40.277.7
        Version 12.41.296.5
(beta)  Version 13.42.319.6
(canary) Version 14.42.334.0

Available - Crosswalk-lite - ionic browser add crosswalk-lite
(canary) Version 10.39.234.
(canary) Version 10.39.236.1

Available - Browser (default) - ionic browser revert android
Not Available Yet - GeckoView
```

你将看到，目前还不支持*WKWebView*和*UIWebView*。但是对于Android应用，你可以使用Crosswalk。想要给已有项目（Example 3）添加Crosswalk的话，运行：

```
ionic browser add crosswalk
```

一旦浏览器添加成功，可以查看*ionic.project*文件进行验证。
想要还原默认的浏览器的时候，运行：

```
ionic browser revert android
```

Ionic lib

可以通过以下命令更新到最新的Ionic库版本：

```
ionic lib update
```

> 也可以传入指定的版本号：**ionic lib update -v 1.0.0-rc.1**

Ionic state

你可以通过Ionic state任务来管理项目的状态。 这么说吧， 你正在你的Ionic应用中添加一些插件和平台以进行测试； 但是你不想在他们测试失败的之后还使用他们。 在这样的情况下，你就可以用save和restore任务了。
你可以通过添加*--nosave*标记来避免将这些插件和平台保存到*package.json*：

```
ionic plugin add cordova-plugin-console --nosave
```

在插件和平台（这些插件和平台添加的时候使用了*--nosave*标记）测试正常之后。如果此时想要将他们添加到*package.json*的话，运行如下命令：

```
ionic state save
```

这个命令将会查找你已经安装的插件和平台，然后将所需的信息添加到*package.json*文件。
可以通过添加*--plugins*标记或者*--platform**标记来指定只保存插件或者平台。
如果添加的插件和平台不能如预期运行，那么可以通过以下命令还原项目到之前的状态：

```
ionic state reset
```

如果想恢复应用的Cordova插件和平台的话，可以在*package.json*里面更新然后运行：

```
ionic state restore
```

> reset任务删除platforms和plugins文件夹然后重新安装，restore只是恢复platforms和plugins文件夹里面缺失的平台和插件。

Ionic ions

根据ions CLI：

> "Ionic ions are a curated collection of useful addons, components, and ux interactions for extending ionic." Ionic ions是一个设计好的用于扩展ionic的插件，组件以及用户交互的集合。

截至本书编写的今日为止，还只有4个ions。可以通过以下命令查看ions列表：

```
ionic ions
```

将会出现如下选项：

```
Header Shrink ........ 'ionic add ionic-ion-header-shrink'
A shrinking header effect like Facebook's

Android Drawer ....... 'ionic add ionic-ion-drawer'
Android-style drawer menu

iOS Rounded Buttons .. 'ionic add ionic-ion-ios-buttons'
iOS "Squircle" style icons

Swipeable Cards ...... 'ionic add ionic-ion-swipe-cards'
Swiping interaction as seen in Jelly

Tinder Cards ......... 'ionic add ionic-ion-tinder-cards'
Tinder style card swiping interaction
```

你可以通过输出结果里面展示的*Add*任务添加ion。一旦添加完成，我们就可以去*www/lib/*下对应的ion文件夹内查看此组件。
例如，当我们新建一个空白项目（example4）的时候，我们可以通过以下命令添加Swipe Card：

```
ionic add ionic-ion-swipe-cards
```

此时，进入*www/lib/ionic-ion-swipe-cards*文件夹就可以找到对应的bower组件来。
在*example14*文件夹内，可以找到更多的设置信息。
可以通过以下命令移除ion：

```
ionic rm ionic-ion-swipe-cards
```

> *ionic add*和*ionic rm*任务也可以用作添加和移除bower包。

Ionic resources

当你添加一个新平台的时候，默认会为新平台创建一个*resources*文件夹，且包含图标和截屏。这些图标和截屏都是默认的图片。如果想为项目使用你自己的标志或者图片，你只需要运行Ionic resources任务就可以了。
这个任务将会在*resources*文件夹内查找一个名为*icon.png*的图片用以为此操作系统的所有设备制作图标，*resources*文件夹里面的*splash.png*用以为此操作系统的所有设备生成截屏。
你可以将这两个图片替换为你自己特色的图片，然后运行：

```
ionic resources
```

如果只想转换图标的话，传入*-i*标记，如果只想转换截屏的话，传入*-s*标记。

> 同时，你也可以使用*.png*, *.psd*（样本范例：http://code.ionicframework.com/resources/icon.psd 与 http://code.ionicframework.com/resources/splash.psd），或者 *.ai*文件来生成图标。更多信息，参考：http://blog.ionic.io/automating-icons-and-splash-screens/

Ionic server, emulate, run

Ionic提供了简单的方法用于在浏览器，模拟器以及设备上运行应用。这三个方法都有很多有用的选项。
如果想要像在真实设备上那样在模拟器上实时重载，那么在调试的时候，使用*-l*标记实现实时重载，使用*-c*激活JavaScript控制台错误输出。 这是在Ionic CLI中至今使用最好和最广的工具方法。这个任务为了节省了大量的调试时间：

```
ionic serve -l -c
ionic emulate -l -c
ionic run -l -c
```

以下是用在*ionic serve*中的可用选项：

```
serve [options] ............................ Start a local development server for app dev/testing
      [--consolelogs|-c] ................... Print app console logs to Ionic CLI
      [--serverlogs|-s] .................... Print dev server logs to Ionic CLI
      [--port|-p] .......................... Dev server HTTP port (8100 default)
      [--livereload-port|-r] ............... Live Reload port (35729 default)
      [--nobrowser|-b] ..................... Disable launching a browser
      [--nolivereload|-d] .................. Do not start live reload
      [--noproxy|-x] ....................... Do not add proxies
      [--address] .......................... Use specific address or return with failure
      [--all|-a] ........................... Have the server listen on all addresses (0.0.0.0)
      [--browser|-w] ....................... Specifies the browser to use (safari, firefox, chrome)
      [--browseroption|-o] ................. Specifies a path to open to (/#/tab/dash)
      [--lab|-l] ........................... Test your apps on multiple screen sizes and platform types
      [--nogulp] ........................... Disable running gulp during serve
      [--platform|-t] ...................... Start serve with a specific platform (ios/android)
```

如果你的应用在Android和iOS有不同的显示效果的时候，可以通过以下命令同时测试：

```
ionic serve -l
```

你可以自己学着探索一下上面提供的选项。当使用ionic run和emulate的时候，可以使用以下选项：

```
run [options] <PLATFORM> ...................  Run an Ionic project on a connected device
    [--livereload|-l] ....................  Live reload app dev files from the device (beta)
    [--port|-p] ..........................  Dev server HTTP port (8100 default, livereload req.)
    [--livereload-port|-r] ...............  Live Reload port (35729 default, livereload req.)
    [--consolelogs|-c] ...................  Print app console logs to Ionic CLI (livereload req.)
    [--serverlogs|-s] ....................  Print dev server logs to Ionic CLI (livereload req.)
    [--debug|--release]
    [--device|--emulator|--target=FOO]

emulate [options] <PLATFORM> ..............  Emulate an Ionic project on a simulator or emulator
        [--livereload|-l] ................  Live reload app dev files from the device (beta)
        [--port|-p] ......................  Dev server HTTP port (8100 default, livereload req.)
        [--livereload-port|-r] ...........  Live Reload port (35729 default, livereload req.)
        [--consolelogs|-c] ...............  Print app console logs to Ionic CLI (livereload req.)
        [--serverlogs|-s] ................  Print dev server logs to Ionic CLI (livereload req.)
        [--nohooks|-n] ...................  Do not add default Ionic hooks for Cordova
        [--debug|--release]
        [--device|--emulator|--target=FOO]
```

自述性非常的好。

Ionic upload和share

可以通过如下命令将你的Ionic项目上传到你的Ionic.io账户：

```
ionic upload
```

> 使用这个功能需要一个Ionic.io账户

应用上传成功之后，就可以通过 https://apps.ionic.io/apps 来查看新传的app。可以通过share命令来与他人分享这个应用，分享需要传入分享对象的邮件地址；例如：

```
ionic share arvind.ravulavaru@gmail.com
```

Ionic view

可以通过Ionic view在设备上预览你的应用。一旦应用上传到你的Ionic.io账户，你可以在Android或者iOS是下载Ionic View应用然后在设备上预览应用。

> 更多Ionic View的信息，参考：http://view.ionic.io/

Ionic help和docs

任何时间点，你都可以通过如下命令查看所有的Ionic CLI任务列表：

```
ionic -h
```

可以通过如下命令打开ionic文档：

```
ionic docs
```

想要查看可用文档，运行：

```
ionic docs ls
```

想要打开指定文档（如*ionicBody*），运行：

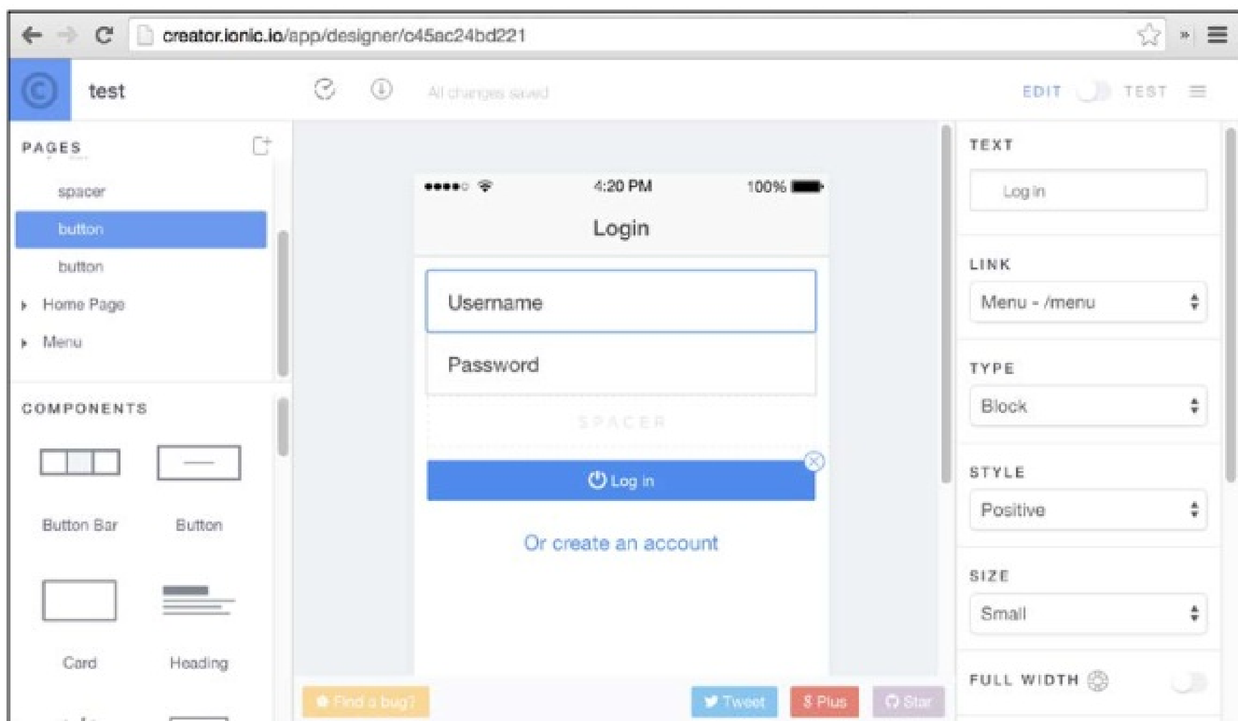
```
ionic docs ionicBody
```

Ionic Creator

Ionic Creator是一个用来轻松搭建Ionic UI的工具。 导航到 http://creator.ionic.io/ 开始使用此工具。 使用这个工具需要一个Ionic.io账号。

有了Ionic Creator，你可以通过拖动放置Ionic组件来创建应用原型。 这个应用是存放在云端的，你可以随时访问并更改。

设计app中的Ionic Creator截屏：



完成设计之后，可以通过以下三种方法来下载你的app：

- 第一个，使用Ionic CLI： **ionic start [appName] creator:c45ac24bd221**
- 第二个，下载项目的ZIP文件

- 最后个，下载纯HTML

可以通过点击左上角的 ⓘ （导出） 按钮看到这些选项。

> 更多关于Ionic Creator信息，参考： http://thejackalofjavascript.com/ionic-creator-beta/

Ionic.io 应用

你可以在 https://apps.ionic.io/apps 创建和管理你的Ionic应用。 在之前的任务中，我们使用的app ID都是我们通过 https://apps.ionic.io/apps 接口创建项目的时候生成的app ID。
可以通过 https://apps.ionic.io/apps 页面上的**New App**按钮来创建一个新的app。 app创建完成之后，点击app的名字就可以看到app的详细信息了。
在app详细信息页面上点击**Settings**可以更新app设置。

> 可以此页面上查看Ionic应用的设置：http://docs.ionic.io/v1.0/docs/io-quick-start 截至本书编写之日，使用Ionic Creator创建的app都没有出现在 https://apps.ionic.io/apps

Ionic Push

可以通过添加Push插件（https://github.com/phonegap-build/PushPlugin ）并进行配置来给你的Ionic应用添加推送消息。 也可以通过使用Ionic的推送模板来达成：

```
ionic add ionic-service-core
ionic add ionic-service-push
ionic start myPushApp push
cd myPushApp
ionic plugin add https://github.com/phonegap-build/PushPlugin.git
ionic upload
```

现在，返回*app.ionc.io*页面的时候，点击之前应用的设置。在*www/js/app.js*的*config*部分，你可以看到之前在应用设置页面做的改动：

```
.config(['$ionicAppProvider', function($ionicAppProvider) {
    // Identify app
    $ionicAppProvider.identify({
        // The App ID for the server
        app_id: 'YOUR_APP_ID',
        // The API key all services will use for this app
        api_key: 'YOUR_PUBLIC_API_KEY'
    });
}])
```

然后，你可以按照Android推送设置引导（(http://docs.ionic.io/v1.0/docs/push-android-setup） 或者iOS推送设置引导（http://docs.ionic.io/v1.0/docs/push-ios-setup） 实现推送消息。

> 更多关于给应用整合推送消息的详细信息，请参考： http://docs.ionic.io/v1.0/docs/push-from-scratch

Ionic Deploy

Ionic Deploy是一个Ionic.io服务。 Ionic可以让你部署新的变更而不用调教的App Store。 这样就为开发者想用户推送新的变更节省了大量的时候。

> 不需要更新安装包的的变更只能被推送-例如，HTML，CSS,JavaScript和图片资源

截至本书编写之日，Ionic Deploy还是Alpha状态。

> 更多关于Ionic deploy的信息，参考： http://blog.ionic.io/announcing-ionic-deploy-alpha-update-your-appwithout-waiting/ 和 http://docs.ionic.io/v1.0/docs/deploy-from-scratch

Ionic Vagrant box

如果你的项目有多个成员，每个成员使用不用环境开发Ionic应用，可以通过Ionic Vagrant box来统一大家的开发环境。

> 如果不懂Vagrant，查阅：http://vagrantup.com 更多关于ionic-box的信息，参考：https://github.com/driftyco/ionic-box

Ionic Sublime Text插件

如果你是Sublime用户，想自动补齐Ionic脚本，那么可以安装以下包：

- Ionic snippets: https://packagecontrol.io/packages/Ionic%20Snippets
- Ionic Framework snippets: https://packagecontrol.io/packages/Ionic%20Framework%20Snippets
- Ionic Framework Extended Autocomplete: https://packagecontrol.io/packages/Ionic%20Framework%20Extended%20Autocomplete

总结

完成这些之后，我们就正式完结 *Learning Ionic* 了。 希望你能够像我这么卖力的讲解Ionic这样卖力的去喜欢Learning Ionic。

我的推特： https://twitter.com/arvindr21 我的Github： https://github.com/arvindr21 （羞羞哒的公开在此，原文：shameless publicity :D）

青山不改，绿水长流，再见，谢谢！

索引